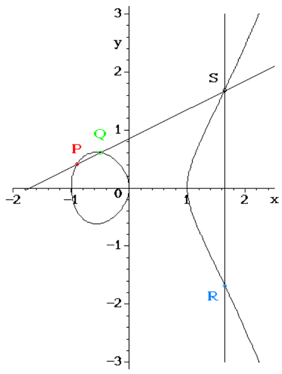

Universität Paderborn

STUDIENARBEIT

Laufzeitanalyse und -vergleich verschiedener Algorithmen zur schnellen skalaren Multiplikation auf elliptischen Kurven

Vorgelegt von:
Tim Süß

Betreuer:
Prof. Dr.rer.nat. Johannes Blömer
Dipl.-Math. Dipl.-Inf. Martin Otto

Paderborn, 5. September 2005

Tim Süß
Hermannstraße 16
33102 Paderborn

Erklärung:
Ich versichere, dass ich diese Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der hier angegebenen Quellen angefertigt habe und diese Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Paderborn, den 5. September 2005 Tim Süß

Inhaltsverzeichnis

Zusammenfassung

Elliptische Kurven werden zur asymmetrischen Kryptographie genutzt. Hierbei werden im Gegensatz zu herkömmlichen Systemen kürzere Schlüssel verwendet, was unter anderem der Nachrichtenexpansion entgegenwirkt. In dieser Arbeit werden elliptische Kurven in verschiedenen Koordinatensystemen über unterschiedlichen Körpern, Algorithmen zur schnellen skalaren Multiplikation und Algorithmen zur modularen Reduktion vorgestellt und miteinander verglichen. Ein weiterer Teil dieser Studienarbeit ist es, die Grundlagen einer C++-Bibliothek und ein Analysetool zu schaffen, um Punktoperationen auf elliptischen Kurven zu berechnen und zu visualisieren.

1 Einleitung

Elliptische Kurven bieten die gleiche Sicherheit wie herkömmliche Verfahren, beispielsweise RSA über $\mathbb{Z}_N$, bei geringerer Schlüssellänge.
Im Allgemeinen lassen sich elliptische Kurven über jedem Körper definieren. In der Kryptographie werden Kurven über endlichen Körpern verwendet.

Definition 1.1 *Sei $p \in \mathbb{Z}$ eine Primzahl, dann bezeichnet $\mathbb{F}_p = \{0, 1, \cdots, p-1\}$ einen endlichen Primkörper mit p Elementen. $\mathbb{F}_p$ ist eindeutig bis auf Isomorphie.*

Definition 1.2 *Sei $f \in \mathbb{F}_p[x]$ ein irreduzibles Polynom vom Grad q, dann bezeichnet $\mathbb{F}_{2^q} = \mathbb{F}_2/ < f >$ den endlichen Körper der Charakteristik 2 mit 2^q Elementen. $\mathbb{F}_{2^q}$ ist eindeutig bis auf Isomorphie.*

Diese Kurven werden über das Weierstraß-Polynom definiert, dabei gilt: $f(x, y) = x^3+ax+b-y^2$ bei Primkörpern und $f(x, y) = x^3+ax^2+b-y^2-yx$ bei Körpern der Charakteristik 2. Allgemein bilden elliptische Kurven additive Gruppen und die Sicherheit basiert auf dem vermutlich schwer zu lösenden Problem des diskreten Logarithmus. Da die in dieser Arbeit betrachteten elliptischen Kurven additive Gruppen sind, wird immer die additive Schreibweise verwendet, wenn es nicht anders angegeben ist.
Die Studienarbeit besteht aus zwei Teilen, die aufeinander aufbauen. Im theoretischen Teil werden die Laufzeiten der unterschiedlichen Teilalgorithmen, welche bei der Implementierung der Bibliothek eingesetzt werden, analysiert. Hierbei werden die benötigten Operationen in den verschiedenen Koordinatensystemen gegenübergestellt. Es wird analysiert, welche Kosten die Berechnung eines multiplikativen Inversen haben darf, damit Punktoperationen in der affinen Ebene die gleichen Kosten habe wie im projektiven Raum oder im Jacobi-Raum. Insbesondere werden die Kostenunterschiede zwischen Verdopplung und Addition betrachtet. Die Anzahl der Operationen bei Verwendung verschiedener Reduktionsalgorithmen wird verglichen. Algorithmen wie die Barrett-Reduktion und die Montgomery-Reduktion sind effizienter als die Standardverfahren zur modularen Reduktion, jedoch bringen auch diese Algorithmen verschiedene Vor- und Nachteile mit sich. Beide Verfahren benötigen diverse Vorberechnungen und das Ergebnis der Montgomery-Reduktion muss vor einer Weiterverwendung in eine andere Form gebracht werden.

Für die Einsetzbarkeit elliptischer Kurven in der Kryptographie sind verwendbare Algorithmen zur skalaren Multiplikation sowie deren Laufzeiten von Interesse; diese werden im theoretischen Teil dieser Arbeit miteinander verglichen. So werden einfache skalare Multiplizierer durch wiederholtes Verdoppeln betrachtet, aber auch Multiplizierer, die den Skalar in eine andere Form bringen und diese neue Darstellung nutzen um die skalare Multiplikation schneller zu berechnen.
Der zweite Teil der Studienarbeit besteht darin, die Grundlagen einer Algorithmen-Bibliothek für elliptische Kurven zu schaffen. Die Bibliothek wird in der Programmiersprache „C++" implementiert und kann als Grundlage weiterer Studienarbeiten dienen. Hierbei wird die von Victor Shoup entwickelte „NTL" (Number Theory Library) verwendet. Die entwickelte Bibliothek beinhaltet elliptische Kurven über Primkörpern und Körpern der Charakteristik 2 in verschiedenen Koordinatendarstellungen. Weiterhin sind verschiedene Algorithmen zur skalaren Multiplikation, Algorithmen zur pseudozufälligen Punktgenerierung auf einer festen Kurve und Funktionen zur modularen Reduktion und Multiplikation enthalten. Zur Untersuchung der Algorithmen und Laufzeiten wird ein Analysetool implementiert. Die Bibliothek sowie die grafische Analyseoberfläche werden nur für Linux kompiliert. Durch die verwendeten plattformunabhängigen Bibliotheken, wie „QT" und „NTL", sollte eine Portierung auf andere Systeme möglich sein.
Im ersten Abschnitt der Ausarbeitung wird eine kurze Einführung in elliptische Kurven gegeben. Im zweiten Abschnitt werden verschiedene Algorithmen vorgestellt, die im dritten Teil miteinander verglichen werden. Im vierten Abschnitt wird ein Zwischenfazit gezogen, indem die bis dorthin gezeigt Ergebnisse zusammengefasst werden. Im fünften Teil werden Besonderheiten bei der Implementierung der Bibliothek betrachtet.

2 Elliptische Kurven

Elliptische Kurven sind über beliebigen Körpern definiert und werden über das Weierstraß-Polynom, $f(x,y) = x^3 - a_1xy + a_2x^2 - a_3y + a_4x + a_6 - y^2$ bestimmt. Diese Kurven bilden Punktmengen auf dem Graph $E: \ y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$, für den gilt: $f(x,y) = 0$; d.h. die Punkte einer solchen Kurve sind die Nullstellen eines Weierstraß-Polynoms. Die Diskriminante einer elliptischen Kurve muss ungleich Null sein, da ihr Graph in keinem Punkt singulär sein darf. Die Diskriminante eines Polynom gibt an, wieviele Nullstellen das Polynom hat; ist der Wert der Diskriminante gleich Null hat das Polynom eine mehrfache Nullstelle. In der Kryptographie werden Kurven über Primkörpern und Körpern der Charakteristik 2 verwendet. Bei Kurven über Primkörpern wird das Weierstraß-Polynom so umgestellt, dass die Koeffizienten $a_1, a_2, a_3 = 0$ sind; $y^2 = x^3 + a_4x + a_6$. Hier muss gelten: $4 \cdot a_4^3 + 27 \cdot a_6^2 \neq 0$, damit die Kurve in keinem Punkt singulär ist. Werden Körper der Charakteristik 2 verwendet, wird das Weierstraß-Polynom so umgestellt, dass die Koeffizienten $a_2, a_3, a_4 = 0$ sind und der Koeffizient $a_1 = 1$ ist; $y^2 + xy = x^3 + a_2x^2 + a_6$. Hier muss $a_6 \neq 0$ gelten. Um elliptische Kurven als Gruppe zu definieren, muss auf jeder dieser Kurven ein Punkt liegen der das neutrale Element der Addition ist. In der affinen Ebene liegt dieser Punkt im Unendlichen und wird als $\mathcal{O}$ bezeichnet. Im projektiven Raum wird $\mathcal{O}$ durch den Punkt $(x,y,z) = (0,1,0)$ repräsentiert, im

Jacobi-Raum durch den Punkt $(x, y, z) = (1, 1, 0)$.
Addiert wird mit dem Lineal. Dabei sind P und Q beliebige Punkte auf der Kurve E, für die gilt $P \neq Q$. Eine Gerade, die durch die beiden Punkte verläuft, schneidet die Kurve genau ein weiteres Mal in einem Punkt. Dieser weitere Schnittpunkt wird als R bezeichnet und die Verknüpfung des Punktes P mit dem Punkt Q sei gekennzeichnet durch $\circ$. Um die Summe von P, Q zu erhalten, muss R entlang der x-Achse gespiegelt werden: $P + Q = -R = -(P \circ Q)$.

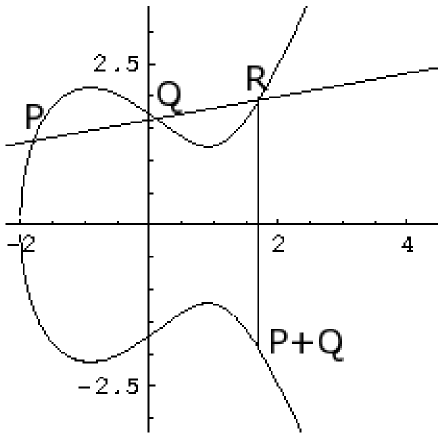

Abbildung 1: $y^2 = x^3 - 2x + 3,\ P + Q = -R$

Sei $P = Q$, dann schneidet die Tangente, die an dem Punkt P anliegt, die Kurve genau ein weiteres Mal in dem Punkt R. Die Spiegelung des Punktes R entlang der x-Achse ist die Verdopplung des Punktes P.

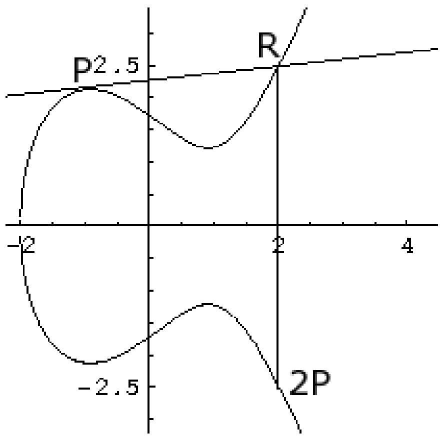

Abbildung 2: $y^2 = x^3 - 2x + 3,\ 2P = -R$

Sei $Q = -P$, so schneidet die Gerade, die durch P und Q verläuft, die Kurve in dem Punkt $\mathcal{O}$ im Unendlichen. Die Summe der Punkte ist somit der Punkt im Unendlichen, der Punkt $\mathcal{O}$.

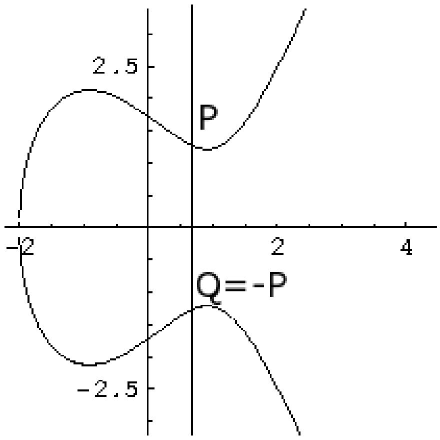

Abbildung 3: $y^2 = x^3 - 2x + 3,\ P + Q = P + (-P) = \mathcal{O}$

Sei $Q = \mathcal{O}$, so schneidet die Gerade, die durch P und $Q = \mathcal{O}$ verläuft die Kurve ein weiteres Mal in $-P$. Die Addition von $\mathcal{O}$ und P ergibt somit wieder P.

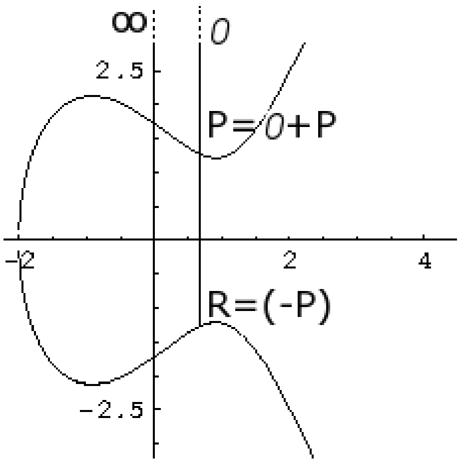

Abbildung 4: $y^2 = x^3 - 2x + 3,\ P + \mathcal{O} = P$

Die Punktaddition und die Punktverdopplung kann auch algebraisch und in verschiedenen Koordinatensystemen berechnet werden, diese Algorithmen werden ab Abschnitt 3.2 beschrieben.

2.1 Gruppengesetze auf elliptischen Kurven

Definition 2.1 *Eine abel'sche Gruppe ist eine Gruppe, für die das Kommutativgesetz gilt.*

Elliptische Kurven beschreiben eine additive abel'sche Gruppe. Es gelten die Gruppengesetze:

1. Neutrales Element:
Es existiert ein $E \in G$, so dass für jedes $x \in G$ gilt: $x \circ E = x$

2. Inverses Element:
Zu jedem $x \in G$ existiert ein $x' \in G$, so dass $x \circ x' = E$ gilt.

3. Assoziativität:
Für jedes Tripel $x, y, z \in G$ gilt, $(x \circ y) \circ z = x \circ (y \circ z)$.

4. Kommutativität:
Für jedes Paar $x, y \in G$ gilt, $x \circ y = y \circ x$

Beweis zu 1: Sei P ein beliebiger Punkt einer elliptischen Kurve, dann gilt:
$\mathcal{O} \circ P = R = -P \Rightarrow \mathcal{O} + P = -(-P) = P$

□

Beweis zu 2: Die Korrektheit dieser Aussage folgt aus 1.
Sei P ein beliebiger Punkt einer elliptischen Kurve, dann gilt:
$\mathcal{O} = (P + \mathcal{O}) + (-P) = P + (-P)$

□

Beweis zu 3: Aufgrund der Länge des vollständigen Beweises wird hier nur auf die Literatur verwiesen [11].
Beweis zu 4: Seien P, Q beliebige Punkte einer elliptischen Kurve, für die gilt: $P \neq Q$. Zu zeigen ist: $P + Q = Q + P$. Die Korrektheit dieser Aussage folgt direkt aus der Definition von $P \circ Q$. Die Reihenfolge in der die Gerade die beiden Punkte P und Q durchläuft, verändert nicht den dritten Schnittpunkt mit der Kurve und damit auch nicht das Ergebnis [1].

□

3 Algorithmische Grundlagen

In diesem Abschnitt werden die Algorithmen vorgestellt, die bei der Implementierung umgesetzt werden. Da in der Kryptographie mit elliptischen Kurven, kurz ECC, immer Kurven über endlichen Körpern verwendet werden und das Einsatzgebiet nicht im Voraus bekannt ist, ist die Effizienz der verwendeten Algorithmen von entscheidender Wichtigkeit. Beispielsweise haben Smartcards sehr beschränkten Speicherplatz und ein langsames Rechenwerk.

3.1 Reduktionsalgorithmen

Die Algorithmen zur modularen Reduktion, die hier vorgestellt werden, sind effizienter als das herkömmliche Verfahren zur Bestimmung des modularen Restes in einem endlichen Körper. Bei dem Standardalgorithmus ist eine Division mit Rest, dabei muss ganzzahlig dividiert, multipliziert und subtrahiert werden.

Algorithmus 1 Standardreduktion

Eingabe: x, p
Ausgabe: z $\equiv$ x $\pmod{p}$, wobei $0 \leq z < p$

1: $\mathtt{q} \leftarrow \lfloor \mathtt{x/p} \rfloor$
2: $\mathtt{r} \leftarrow \mathtt{q} \cdot \mathtt{p}$
3: $\mathtt{z} \leftarrow \mathtt{x} - \mathtt{r}$
4: **Return z**

In dieser Studienarbeit wird immer angenommen, dass nicht direkt auf den Rest einer Division zugegriffen werden kann. Statt einer ganzzahligen Division können auch kostengünstigere Algorithmen zur modularen Reduktion verwendet werden.
Ist der gegebene Körper prim und wird die Barrett-Reduktion oder die Montgomery-Reduktion verwendet, so kann der Rest einer ganzen Zahl durch wenige Bitoperationen bestimmt werden. Bei der Barrett- und Montgomery-Reduktion müssen für einen festen Körper einige wenige Werte einmalig vorberechnet werden. Diese Werte können gespeichert und wiederverwendet werden, wodurch eine schnelle Reduktion möglich ist.

3.1.1 Barrett-Reduktion

Die Barrett-Reduktion ist eine Möglichkeit, eine schnelle modulare Reduktion $z \equiv x \pmod{p}$ zu berechnen, wobei $0 \leq z < p$ ist.

Definition 3.1 *Sei p eine feste Primzahl. Der Wert t sei definiert als der Exponent der Basis b, für den gilt:*

$$b^{t-1} \leq p < b^t$$

Definition 3.2 *Die Basis in der eine ganze Zahl p dargestellt wird sei definiert als b, sodass gilt:*

$$p = \sum_{j=0}^{t-1} a_j b^j$$

Der Einsatz des Barrett-Algorithmus ist dann sinnvoll, wenn der Modul p fest ist und viele Reduktionen in einem endlichen Körper berechnet werden müssen. Da $\mu = \lfloor b^{2t}/p \rfloor$ in Abhängigkeit von dem Modul p durch eine ganzzahlige Division berechnet wird und dieser Wert für jede Reduktion gleich ist, kann μ gespeichert und wiederverwendet werden. Die ganzzahlige Division kann in $O(n^2)$ Bitoperationen berechnet werden [7]. Damit der Algorithmus effizient eingesetzt werden kann, muss der zu reduzierende Wert kleiner als das Quadrat des Moduls sein. Die Barrett-Reduktion wird durch folgenden Algorithmus berechnet [5].

Algorithmus 2 Barrett-Reduktion

Eingabe: x, p, μ, sodass x$< b^{2t}$, $b^{t-1} < p \leq b^t$ und $\mu = \lfloor b^{2t}/p \rfloor$

Ausgabe: z $\equiv$ x $\pmod{p}$, wobei $0 \leq$ z $< p$

```
q0 ← ⌊x/b^(t-1)⌋
q  ← ⌊(μq0)/b^(t+1)⌋
r1 ← x (mod b^(t+1)), r2 := qp (mod b^(t+1))
z  ← r1 − r2
If z < 0 then z ← z + b^(t+1)
While z ≥ p do z ← z − p
Return z
```

Satz 3.1 *Der Algorithmus zur Barrett-Reduktion berechnet die modulare Reduktion* $z \equiv x \pmod{p}$, $0 \leq z < p$.

Beweis: Für das zu reduzierende x gilt: $x = Q \cdot p + R,\ 0 \leq R < p$.
Das q, das in den ersten beiden Schritten des Algorithmus bestimmt wird, ist eine Abschätzung von $Q = \frac{x-R}{p}$. Zuerst wird gezeigt, das $Q - 2 \leq q \leq Q$.
1. Zu zeigen $q \leq Q$. Es gilt:

$$\begin{aligned} Q = \tfrac{x-R}{p} &= \left\lfloor \tfrac{x}{p} \right\rfloor = \left\lfloor \tfrac{x}{p} \cdot \tfrac{b^{2t}}{b^{2t}} \right\rfloor = \left\lfloor \tfrac{x}{b^{t-1}} \cdot \tfrac{b^{2t}}{p} \cdot \tfrac{1}{b^{t+1}} \right\rfloor \\ &\geq \left\lfloor \left\lfloor \tfrac{x}{b^{t-1}} \right\rfloor \cdot \left\lfloor \tfrac{b^{2t}}{p} \right\rfloor \cdot \tfrac{1}{b^{t+1}} \right\rfloor = q \end{aligned}$$

2. Zu zeigen $q \geq Q - 2$. Es gilt:

$$\begin{aligned} q &= \left\lfloor \left\lfloor \frac{x}{b^{t-1}} \right\rfloor \cdot \left\lfloor \frac{b^{2t}}{p} \right\rfloor \cdot \frac{1}{b^{t+1}} \right\rfloor \geq \left\lfloor \left(\frac{x}{b^{t-1}} - 1 \right) \cdot \left(\frac{b^{2t}}{p} - 1 \right) \cdot \frac{1}{b^{t+1}} \right\rfloor \\ &= \left\lfloor \left(\frac{x \cdot b^{2t}}{b^{t-1} \cdot p} - \frac{x}{b^{t-1}} - \frac{b^{2t}}{p} + 1 \right) \cdot \frac{1}{b^{t+1}} \right\rfloor \\ &= \left\lfloor \left(\frac{x \cdot b^{t+1}}{p} - \frac{x}{b^{t-1}} - \frac{b^{2t}}{p} + 1 \right) \cdot \frac{1}{b^{t+1}} \right\rfloor \\ &= \left\lfloor \frac{x}{p} - \left(\frac{x}{b^{t-1}} + \frac{b^{2t}}{p} - 1 \right) \cdot \frac{1}{b^{t+1}} \right\rfloor \\ &= \left\lfloor \frac{x}{p} - \frac{R}{p} + \frac{R}{p} - \left(\frac{x}{b^{t-1}} + \frac{b^{2t}}{p} - 1 \right) \cdot \frac{1}{b^{t+1}} \right\rfloor \\ &= \left\lfloor \frac{x-R}{p} + \frac{R}{p} - \left(\frac{x}{b^{t-1}} + \frac{b^{2t}}{p} - 1 \right) \cdot \frac{1}{b^{t+1}} \right\rfloor \\ &= \left\lfloor Q + \frac{R}{p} - \left(\frac{x}{b^{t-1}} + \frac{b^{2t}}{p} - 1 \right) \cdot \frac{1}{b^{t+1}} \right\rfloor \end{aligned}$$

Da Q ganzzahlig und ein Summand ist, kann es vor das *floor* geschrieben werden.

$$= Q + \left\lfloor \frac{R}{p} - \left(\frac{x}{b^{t-1}} + \frac{b^{2t}}{p} - 1 \right) \cdot \frac{1}{b^{t+1}} \right\rfloor$$
$$= Q + \left\lfloor \frac{R}{p} - \left(\frac{x}{b^{2t}} + \frac{b^{t-1}}{p} - \frac{1}{b^{t+1}} \right) \right\rfloor$$

Die Werte im *floor* werden zum Minimum abgeschätzt.

$$Q + \Big\lfloor \underbrace{\frac{R}{p}}_{>0,\,<1} - \Big(\underbrace{\frac{x}{b^{2t}}}_{>0,\,<1} + \underbrace{\frac{b^{t-1}}{p}}_{>0,\,<1} - \underbrace{\frac{1}{b^{t+1}}}_{>0,\,<1} \Big) \Big\rfloor$$
$$\geq Q + \lfloor 0 - 1 - 1 - 0 \rfloor = Q - 2$$

Damit ist gezeigt, dass q eine Abschätzung von Q ist, die nur maximal um 2 kleiner ist. Es muss noch gezeigt werden, dass $r_1 - r_2 \equiv R \pmod p$ gilt.
Es gilt:

$$\begin{aligned} r_1 - r_2 \quad &= x \pmod{b^{t+1}} - qp \pmod{b^{t+1}} \equiv x - qp \pmod{b^{t+1}} \\ &= Qp + R - qp \pmod{b^{t+1}} = (Q-q)p + R \pmod{b^{t+1}} \end{aligned}$$

Aus $Q - 2 \leq q \leq Q$, folgt:

$$(Q-q)p + R < b^{t+1} \Rightarrow r_1 - r_2 \equiv R \pmod p$$

□

3.1.2 Montgomery-Reduktion

Die Montgomery-Reduktion ist eine weitere Möglichkeit zur modularen Reduktion, bei der $z \equiv x \pmod p$ berechnet wird , wobei $0 \leq z < p$ gilt. Dieser Algorithmus eignet sich genauso wie das Barrett-Verfahren dazu, viele modularen Reduktionen in einem festen endlichen Körper zu berechnen. Bei diesem Verfahren müssen mehrere Werte im Voraus bestimmt werden. Für den Algorithmus muss eine Basis b gewählt werden, die teilerfremd zu dem Modul p der zu berechnenden Reduktion ist. R sollte so gewählt werden, dass es die nächsthöhere Potenz der Basis b ist, die größer als der Modul ist. Eine derartige Wahl von R eignet sich besonders dazu, ganzzahlige Divisionen und modulare Reduktionen durch Shifts zu berechnen. Die Parameter müssen folgende Form haben:
$GGT(b, p) = 1,\ R = b^t,\ p' = -p^{-1} \pmod R$
$p = (p_{t-1}, p_{t-2}, \ldots, p_1, p_0)_b$,
$x = (x_{2t-1}, x_{2t-2}, \ldots, x_1, x_0)_b$
Die Werte p', R und b werden im Voraus berechnet und gespeichert. Der zu reduzierende Wert muss für einen effizienten Einsatz kleiner als $p \cdot R$ sein. Zur Berechnung einer Montgomery-Reduktion ist folgender Algorithmus gegeben [5].

Der Beweis der Montgomery-Reduktion ist aufwendig, daher wird hier auf die angegebene Literatur verwiesen [13]. Im Weiteren wird von der Korrektheit des Algorithmus ausgegangen.

Algorithmus 3 Montgomery-Reduktion

Eingabe: Eine ganze Zahl $x < pR$

Ausgabe: $z \equiv xR^{-1} \pmod{p}$, mit $0 \leq z < p$

```
For i = 0 to t − 1 do :
    u ← x_i p′ (mod b),
    x ← x + upb^i.
z ← x/R
If z ≥ p then z ← z − p
Return z
```

3.1.3 Montgomery-Multiplikation

Der Montgomery-Multiplikationsalgorithmus berechnet das modulare Produkt zweier ganzer Zahlen x, y im Körper $\mathbb{F}_p$, ohne diese vorher vollständig miteinander zu multiplizieren.

Definition 3.3 $[X]_R = x \cdot R \pmod{p}$ *bezeichnet die Montgomerydarstellung einer ganzen Zahl* x*.*

Definition 3.4 *Sei* $[X]_R$ *eine ganze Zahl in Montgomerydarstellung, dann bezeichen* $[X_i]_R$ *die* i*-te Stelle von* $[X]_R$ *in der Basis* b*.*

$[X]_R, [Y]_R$ seien die Montgomerydarstellungen der Zahlen x, y. R ist hierbei wieder die nächsthöhere Potenz der Basis b, die größer als p ist; es gilt: $b^{t-1} < p < b^t$. Der Algorithmus liefert das Ergebnis in Montgomerydarstellung, $[X]_R [Y]_R R^{-1} = xR{\cdot}yR{\cdot}R^{-1} \pmod{p}$. Um dieses Ergebnis in Normaldarstellung zu überführen, muss es mit $R^{-1} \pmod{p}$ multipliziert werden.

Algorithmus 4 Montgomery-Multiplikation

```
[Z]_R := 0
For i = 0 to t − 1 do :
    u ← ([Z_i]_R + [X_i]_R [Y]_R)p′ (mod b),
    [Z]_R ← [Z]_R + ([X_i]_R [Y]_R + up)b^i.
[Z]_R ← [Z]_R /R
If [Z]_R ≥ p then [Z]_R ← [Z]_R − p
Return [Z]_R
```

Satz 3.2 *Zur Eingabe* $[X]_R [Y]_R$ *mit* $0 \leq [X]_R, [Y]_R < p$ *berechnet Algorithmus 4 das Montgomery-Produkt* $[Z]_R := [X]_R [Y]_R$ *mit* $2t(t+1)$ *Multiplikationen von Faktoren aus* $[0, b)$*.*

Beweis: (Beweis entnommen [13]) Um das Produkt $[Z]_R = [X]_R \cdot [Y]_R$, $[Z]_R = z_{t-1} \cdots z_1 z_0$ zu berechnen, wird die Multiplikation und die Montgomery-Reduktion zusammengefasst. Da

$$[X]_R [Y]_R = \left(\sum_{i=0}^{t-1} [X_i]_R \right) [Y]_R = \sum_{i=0}^{t-1} [X_i]_R [Y]_R$$

kann das Produkt $[Z]_R := [X]_R [Y]_R$ wie folgt berechnet werden:

1. $[\mathtt{Z}]_\mathtt{R} := 0$
2. $\mathtt{For\ i = 0\ to\ t-1\ do:}\ [\mathtt{Z}]_\mathtt{R} := [\mathtt{Z}]_\mathtt{R} + [\mathtt{X_i}]_\mathtt{R}\, \mathtt{b^i}\, [\mathtt{Y}]_\mathtt{R}$

Die Werte $z_0, z_1, \cdots, z_i$ bleiben nach der Iteration i unverändert. Bei der Montgomery-Reduktion in Algorithmus 3 hängt der Wert u nur von x_i ab. Bei der Montgomery-Reduktion von $[X]_R\,[Y]_R$ können wir u bereits nach der Zuweisung $[Z]_R := [Z]_R + [X_i]_R\, b^i\, [Y]_R$ bestimmen. Auf diese Weise lassen sich die Schleifen der Multiplikation und der Montgomery-Reduktion zusammenfassen:

1. $[\mathtt{Z}]_\mathtt{R} := 0$
2. $\mathtt{For\ i = 0\ to\ t-1\ do:}$
   (a) $[\mathtt{Z}]_\mathtt{R} := [\mathtt{Z}]_\mathtt{R} + [\mathtt{X_i}]_\mathtt{R}\, \mathtt{b^i}\, [\mathtt{Y}]_\mathtt{R}$
   (b) $\mathtt{u} := [\mathtt{Z_i}]_\mathtt{R}\, \mathtt{p}' \pmod{\mathtt{b}}$
   (c) $[\mathtt{Z}]_\mathtt{R} := [\mathtt{Z}]_\mathtt{R} + \mathtt{upb^i}$

□

3.2 Koordinatensysteme für elliptische Kurven über Primkörpern

Elliptische Kurven über endlichen Primkörpern sind eine abel'sche Gruppe. Das neutrale Element dieser Gruppe, der Punkt im Unendlichen, wird mit $\mathcal{O}$ bezeichnet. Die verschiedenen Koordinatensysteme, in denen elliptische Kurven dargestellt werden, bringen für die Additionsalgorithmen unterschiedliche Vor- und Nachteile. Punktadditionen in der affinen Ebene benötigen eine kostenintensive Inversion, aber wenige arithmetische Operationen. Die Inversion wird mit dem erweiterten euklidischen Algorithmus bestimmt und benötigt $O(n^3)$ Bitoperationen. Punktadditionen im projektiven Raum und im Jacobi-Raum benötigen keine Inversion, dafür aber mehr arithmetische Additionen und Multiplikationen.

3.2.1 Affine Koordinaten

Die Punkte einer elliptischen Kurve in der affinen Ebene werden über das Weierstraß-Polynom $f(x,y) = x^3 + ax + b - y^2,\ a,b \in \mathbb{F}_p$ bestimmt. Ein Punkt P sei definiert durch seine Koordinaten (x_1, y_1), ein Punkt Q durch (x_2, y_2) und die Summe der Punkte $P + Q = S$ durch (x_3, y_3). Sei $P = \mathcal{O}$ und Q eine beliebige Nullstelle der Kurve, so gilt:
$P + Q = \mathcal{O} + Q = Q = S$
Sei P ein beliebiger Punkt der elliptischen Kurve und $Q = -P$, so gilt:
$P + Q = \mathcal{O} = S$
Werden zwei unterschiedliche Punkte addiert, wird folgender Algorithmus verwendet [5]:

$$P, Q \neq \mathcal{O} \text{ und } P \neq \pm Q$$

$$\lambda := \frac{y_2 - y_1}{x_2 - x_1} \tag{1}$$

$$x_3 := \lambda^2 - x_1 - x_2 \tag{2}$$

$$y_3 := \lambda(x_1 - x_3) - y_1 \tag{3}$$

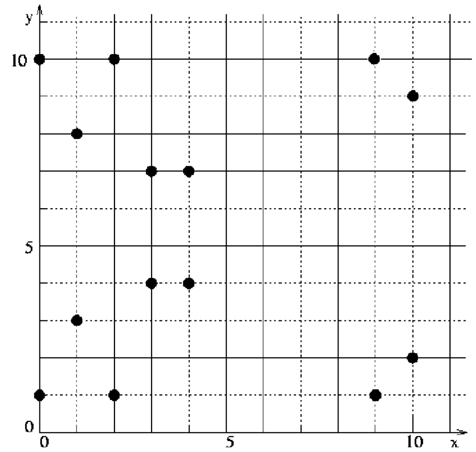

Abbildung 5:
$y^2 = x^3 + 7x + 1 \pmod{11}$

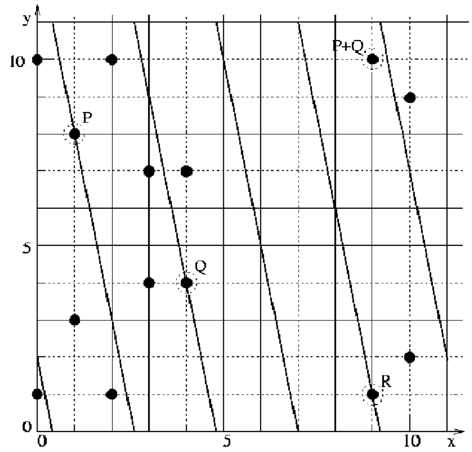

Abbildung 6:
$(1, 8) + (4, 4) = (9, 9) \pmod{11}$

Wird ein Punkt verdoppelt, wird folgender Algorithmus verwendet:

$$P, Q \neq \mathcal{O} \text{ und } P = Q$$

$$\lambda := \frac{3x_1^2 + a}{2y_1} \tag{4}$$

$$x_3 := \lambda^2 - 2x_1 \tag{5}$$

$$y_3 := \lambda(x_1 - x_3) - y_1 \tag{6}$$

3.2.2 Projektive Koordinaten

Das Weierstraß-Polynom im projektiven Raum ist homogen und ist um die Z-Koordinate erweitert: $f(X, Y, Z) = X^3 + aXZ^2 + bZ^3 - Y^2Z,\ a, b \in \mathbb{F}_p$. Die Punktadditionen und Verdopplungen im projektiven Raum, im Gegensatz zur affinen Ebene, benötigen keine Inversion, jedoch mehr Multiplikationen, Additionen und Potenzierungen. Zwei Punkte $P = (X_1, Y_1, Z_1), P' = (X_2, Y_2, Z_2) \in E_{\mathbb{F}_p}$ sind gleich, wenn ein $t \in \mathbb{F}_p$ existiert, sodass $(tX_1, tY_1, tZ_1) = (X_2, Y_2, Z_2)$ gilt. Das neutrale Element der Addition $\mathcal{O}$, ist im projektiven Raum als $(x, y, z) = (0, 1, 0)$ definiert. Die Projektion eines Punktes in die affine Ebene wird für $Z \neq 0$ mittels $x = XZ^{-1},\ y = YZ^{-1}$ berechnet. Die Punktadditionen und Punktverdopplungen werden mit folgenden Algorithmen berechnet [5].

Addition
$P \neq Q$ und $P, Q \neq \mathcal{O}$

$$
\begin{aligned}
u &:= Y_2 Z_1 - Y_1 Z_2 && (7)\\
v &:= X_2 Z_1 - X_1 Z_2 && (8)\\
A &:= (u^2 Z_1 Z_2 - v^3) - (2v^2 X_1 Z_2) && (9)\\
X_3 &:= vA && (10)\\
Y_3 &:= (v^2 X_1 Z_2 - A)u - (v^3 Y_1 Z_2) && (11)\\
Z_3 &:= v^3 Z_1 Z_2 && (12)
\end{aligned}
$$

Verdopplung
$P = Q$ und $P, Q \neq \mathcal{O}$

$$
\begin{aligned}
w &:= (aZ_1^2) + (3X_1^2) && (13)\\
s &:= Y_1 Z_1 && (14)\\
B &:= X_1 Y_1 s && (15)\\
h &:= w^2 - 8B && (16)\\
X_3 &:= 2hs && (17)\\
Y_3 &:= w(4B - h) - (8Y_1^2 s^2) && (18)\\
Z_3 &:= 8s^3 && (19)
\end{aligned}
$$

3.2.3 Jacobi-Koordinaten

Die Darstellung einer elliptischen Kurve in Jacobi-Koordinaten ist eine Erweiterung der projektiven Darstellung. Der Graph einer Kurve wird über das inhomogene Polynom $f(X, Y, Z) = X^3 + aXZ^4 + bZ^6 - Y^2,\ a, b \in \mathbb{F}_p$ bestimmt. Das neutrale Element der Addition $\mathcal{O}$ ist im Jacobi-Raum als $(x, y, z) = (1, 1, 0)$ definiert. Zwei Punkte $P = (X_1, Y_1, Z_1), P' = (X_2, Y_2, Z_2)$ sind gleich, wenn ein $t \in \mathbb{F}_p$ existiert, sodass $(X_2, Y_2, Z_2) = (t^2 X_1, t^3 Y_1, tZ_1)$ gilt. Die Projektion eines Punktes aus dem Jacobi-Raum in die affine Ebene wird für $Z \neq 0$ mittels $x = XZ^{-2},\ y = YZ^{-3}$ berechnet. Die Addition im Jacobi-Raum benötigt keine Inversion. Die Punktaddition zweier Punkte $P, Q,\ P \neq Q$, benötigt mehr arithmetische Operationen, als die Punktaddition im projektiven Raum. Die Verdopplung eines Punktes benötigt jedoch weniger arithmetische Operationen. Die Punktadditionen und Punktverdopplungen werden mit folgenden Algorithmen berechnet [10].

Addition
$P \neq Q$ und $P, Q \neq \mathcal{O}$

$$u_1 := X_1 Z_2^2 \tag{20}$$
$$u_2 := X_2 Z_1^2 \tag{21}$$
$$s_1 := Y_1 Z_2^3 \tag{22}$$
$$s_2 := Y_2 Z_1^3 \tag{23}$$
$$H := u_2 - u_1 \tag{24}$$
$$r := s_2 - s_1 \tag{25}$$
$$X_3 := (r^2 - H^3) - 2u_1 H^2 \tag{26}$$
$$Y_3 := r(u_1 H^2 - X_3) - s_1 H^3 \tag{27}$$
$$Z_3 := Z_1 Z_2 H \tag{28}$$

Verdopplung
$P = Q$ und $P, Q \neq \mathcal{O}$

$$s := 4X_1 Y_1^2 \tag{29}$$
$$M := 3X^2 + aZ_1^4 \tag{30}$$
$$T := M^2 - 2s \tag{31}$$
$$X_3 := T \tag{32}$$
$$Y_3 := M(s - T) - 8Y_1^4 \tag{33}$$
$$Z_3 := 2Y_1 Z_1 \tag{34}$$

3.3 Koordinatensysteme zu elliptischen Kurven über Körpern der Charakteristik 2

Elliptische Kurven über Körpern der Charakteristik 2 haben die gleichen Gruppeneigenschaften wie Kurven über Primkörpern. Der Polynomkörper wird durch ein irreduzibles Polynom definiert. Die Elemente eines Körpers der Charakteristik 2 sind Polynome der Form: $a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x^1 + a_0 x^0,\ a_i \in \{0, 1\}$, wobei n echt kleiner ist als der Grad des irreduziblen Polynoms, das den Körper $\mathbb{F}_{2^q}$ definiert.

3.3.1 Affine Koordinaten

Elliptische Kurven in der affinen Ebene werden durch das Weierstraß-Polynom $f(X, Y) = X^3 + aX^2 + b - Y^2 - XY$ definiert, wobei $a, b \in \mathbb{F}_{2^q}$ und $b \neq 0$ gilt. Das Inverse eines Punktes $P = (x_1, y_1)$ ist $(x_1, x_1 + y_1) = (-P)$.
Die Algorithmen zur Punktaddition und Punktverdopplung sind [5] entnommen. Addition
$P, Q \neq \mathcal{O}$ und $P \neq Q$

$$\lambda := \frac{y_1 + y_2}{x_1 + x_2} \tag{35}$$
$$x_3 := \lambda^2 + \lambda + x_1 + x_2 + a \tag{36}$$
$$y_3 := (x_1 + x_3)\lambda + x_3 + y_1 \tag{37}$$

Verdopplung
$P = Q$ und $P \neq \mathcal{O}$

$$\lambda := \frac{y_1}{x_1} + x_1 \tag{38}$$
$$x_3 := \lambda^2 + \lambda + a \tag{39}$$
$$y_3 := (x_1 + x_3)\lambda + x_3 + y_1 \tag{40}$$

3.3.2 Projektive Koordinaten

Elliptische Kurven über Körpern der Charakteristik 2 im projektiven Raum werden durch das Polynom $f(X,Y,Z) = X^3 + aX^2Z^2 + bZ^6 - Y^2 - XYZ,\ a,b \in \mathbb{F}_{2^q}$ bestimmt. Um einen Punkt im projektiven Raum zu verdoppeln, muss $d = \sqrt[4]{b} = b^{2^{n-2}}$ bestimmt werden. Für jede elliptische Kurve im projektiven Raum ist d fest und kann bei der Initialisierung einer nicht singulären elliptischen Kurve einmalig bestimmt werden. Das additive Inverse eines Punktes $P = (x_1, y_1, z_1)$ ist $(x_1, (x_1 \cdot z_1 + y_1), z_1) = (-P)$. Die Projektion eines Punktes aus dem projektiven Raum in die affine Ebene wird berechnet durch: $X/Z^2,\ Y/Z^3$ bei $Z \neq 0$. Die Algorithmen zur Punktaddition und Punktverdopplung sind [5] entnommen.

Addition
$P, Q \neq \mathcal{O}$ und $P \neq Q$

$$\lambda_1 := X_1 Z_2^2 \tag{41}$$
$$\lambda_2 := X_2 Z_1^2 \tag{42}$$
$$\lambda_3 := \lambda_1 + \lambda_2 \tag{43}$$
$$\lambda_4 := Y_1 Z_2^3 \tag{44}$$
$$\lambda_5 := Y_2 Z_1^3 \tag{45}$$
$$\lambda_6 := \lambda_4 + \lambda_5 \tag{46}$$
$$\lambda_7 := Z_1 \lambda_3 \tag{47}$$
$$\lambda_8 := \lambda_6 X_2 + \lambda_7 Y_2 \tag{48}$$
$$Z_3 := \lambda_7 Z_2 \tag{49}$$
$$\lambda_9 := \lambda_6 + Z_3 \tag{50}$$
$$X_3 := a Z_3^2 + \lambda_6 \lambda_9 + \lambda_3^3 \tag{51}$$
$$Y_3 := \lambda_9 X_3 + \lambda_8 \lambda_7^2 \tag{52}$$

Verdopplung
$P = Q$ und $P \neq \mathcal{O}$

$$Z_3 := X_1 Z_1^2 \tag{53}$$
$$X_3 := (X_1 + d Z_1^2)^4 \tag{54}$$
$$\lambda := Z_3 + X_1^2 + Y_1 Z_1 \tag{55}$$
$$Y_3 := X_1^4 Z_3 + \lambda X_3 \tag{56}$$

3.4 Skalare Multiplikationsalgorithmen

Die Sicherheit einiger kryptographischer Systeme wie *ElGamal* oder *Diffie-Hellman* basieren auf dem Problem des diskreten Logarithmus. Diese Systeme lassen sich mittels elliptischer Kurven realisieren, wobei ein kürzerer Schlüssel verwendet werden kann als bei RSA. Die Verdopplung eines Punktes P oder die Addition zweier Punkte $P, Q \neq \mathcal{O}$ besteht aus verschiedenen arithmetischen Operationen. Das skalare Produkt eines Punktes P wird mittels Additionskette berechnet. Damit kryptographische Systeme, die auf elliptischen Kurven basieren, in der Praxis verwendet werden können, müssen effiziente Algorithmen zur skalaren Multiplikation eingesetzt werden. Für die Analyse der Algorithmen zur skalaren Multiplikation wird das Hamming-Gewicht des Skalars betrachtet.

Definition 3.5 *Sei k eine ganze Zahl in Binärdarstellung, $k = (k_{n-1} \cdots k_1 k_0)_2$ und s die Anzahl der $k_i \neq 0,\ 0 \leq i < n$. Das Hamming-Gewicht von k sei definiert als $\frac{s}{n}$.*

Definition 3.6 $\mathbb{K}$ *sei definiert als fester Primkörper oder fester Polynomkörper der Charakteristik* 2.

Definition 3.7 *Sei k eine ganze Zahl und P ein Punkt auf einer elliptischen Kurve, dann bezeichnet $[k]\,P$ die skalare Multiplikation von P mit k.*

3.4.1 Multiplikation durch wiederholtes Verdoppeln von rechts nach links

Eine Möglichkeit, einen Punkt P schnell zu multiplizieren, ist ein Multiplizierer durch wiederholtes Verdoppeln. Startet der Algorithmus mit dem niederwertigsten Bit, so wird in jedem Schritt P verdoppelt; wenn in der Binärdarstellung des Skalars eine Eins auftritt, wird eine zusätzliche Addition berechnet.

Algorithmus 5 Rechts-Links-Multiplizierer

Eingabe : $\mathtt{P} \in \mathtt{E}_{\mathbb{K}}$, eine l − Bit lange ganze Zahl $\mathtt{k} = \sum_{j=0}^{l-1} \mathtt{k_j} 2^{\mathtt{j}},\ \mathtt{k_j} \in \{0, 1\}$

Ausgabe : $\mathtt{S} = [\mathtt{k}]\,\mathtt{P}$

1: $\mathtt{S} \leftarrow \mathcal{O}$
2: For $\mathtt{i} = 0$ to $\mathtt{l} - 1$ do :
3: If $\mathtt{k_i} == 1$ then $\mathtt{S} \leftarrow \mathtt{S} + \mathtt{P}$.
4: $\mathtt{P} \leftarrow \mathtt{P} + \mathtt{P}$
5: Return S

Satz 3.3 *Algorithmus 5 berechnet die skalare Multiplikation $[k]\,P$.*

Beweis: Der Beweis folgt durch logischen Schluss.

$$\begin{aligned}
[k]\,P &= \left(k_{l-1}\cdot 2^{l-1}+k_{l-2}\cdot 2^{l-2}+\ldots+k_1\cdot 2^1+k_0\cdot 2^0\right)\cdot P\\
&= k_{l-1}\cdot 2^{l-1}P+k_{l-2}\cdot 2^{l-2}P+\ldots+k_1\cdot 2^1P+k_0\cdot 2^0P\\
&= k_0\cdot 2^0P+k_1\cdot 2^1P+\ldots+k_{l-2}\cdot 2^{l-2}P+k_{l-1}\cdot 2^{l-1}P\\
&= \sum_{j=0}^{l-1}\left(k_j\cdot 2^j\cdot P\right)
\end{aligned}$$

□

Der Rechts-Links-Multiplizierer muss im *worst case* $2l$ Kurvenoperationen berechnen; davon l Punktverdopplungen und l Punktadditionen. Im *average case* liegt das *Hamming-Gewicht* des Faktors bei $\frac{1}{2}$. Es müssen somit im Durchschnitt $\frac{3}{2}l$ Kurvenoperationen berechnet werden (l Punktverdopplungen und $\frac{l}{2}$ Punktadditionen).

3.4.2 Multiplikation durch wiederholtes Verdoppeln von links nach rechts

Der Links-Rechts-Multiplizierer beginnt mit dem höchstwertigsten Koeffizienten der Binärdarstellung des Faktors.

Algorithmus 6 Links-Rechts-Multiplizierer

Eingabe : $\mathtt{P}\in\mathtt{E}_{\mathbb{K}}$, eine l − Bit lange ganze Zahl $\mathtt{k}=\sum_{\mathtt{j}=0}^{\mathtt{l}-1}\mathtt{k_j}2^{\mathtt{j}},\ \mathtt{k_j}\in\{0,1\}$

Ausgabe : $\mathtt{S}=[\mathtt{k}]\,\mathtt{P}$

```
1: S ← 𝒪
2: For i = l − 1 to 0 do by − 1 :
3:     S ← S + S
4:     if k_i == 1 then S ← S + P.
5: Return S
```

Satz 3.4 *Algorithmus 6 berechnet die skalare Multiplikation* $[k]\cdot P$.

Beweis: Der Beweis folgt durch logischen Schluss.

$$[k]\,P=\left(k_{l-1}\cdot 2^{l-1}+k_{l-2}\cdot 2^{l-2}+\ldots+k_1\cdot 2^1+k_0\cdot 2^0\right)\cdot P$$
$$=k_{l-1}\cdot 2^{l-1}P+k_{l-2}\cdot 2^{l-2}P+\ldots+k_1\cdot 2^1P+k_0\cdot 2^0P$$
$$=k_{l-1}\cdot P\cdot 2\cdot 2^{l-2}+k_{l-2}\cdot P\cdot 2^{l-2}+\ldots+k_1\cdot P\cdot 2^1+k_0\cdot P\cdot 2^0$$
$$=\left(\left(k_{l-1}\cdot P\right)\cdot 2+k_{l-2}\cdot P\right)\cdot 2^{l-2}+k_{l-3}\cdot P\cdot 2^{l-3}+\ldots+k_1\cdot P\cdot 2^1+k_0\cdot P\cdot 2^0$$
$$=\left(\left(\left(k_{l-1}\cdot P\right)\cdot 2+k_{l-2}\cdot P\right)\cdot 2+k_{l-3}\cdot P\right)\cdot 2^{l-3}+\ldots+k_1\cdot P\cdot 2^1+k_0\cdot P\cdot 2^0$$
$$=\left(\ldots\left(\left(\left(k_{l-1}\cdot P\right)\cdot 2+k_{l-2}\cdot P\right)\cdot 2+k_{l-3}\cdot P\right)\cdot 2+\ldots+k_1\cdot P\right)\cdot 2+k_0\cdot P$$

□

Es ist zu bemerken, dass der binäre Links-Rechts-Multiplizierer immer eine Addition weniger berechnen muss als der binäre Rechts-Links-Multiplizierer. Die erste Verdopplung (im Algorithmus 6 Zeile 3) muss nicht berechnet werden, da $\mathcal{O}$ verdoppelt wird.

3.4.3 NAF-Multiplizierer

Multiplizierer durch wiederholtes Verdoppeln berechnen in jedem Schleifendurchlauf eine Punktverdopplung; wenn in der Binärdarstellung des Faktors ein Wert ungleich Null auftritt, wird eine weitere Punktaddition berechnet. Um die skalare Multiplikation durch wiederholtes Verdoppeln zu optimieren, muss die Anzahl der Werte ungleich Null minimal sein. Die *Nonadjacent Form*, kurz NAF, bietet diese Eigenschaft.

Definition 3.8 *Sei*

$$k = \sum_{j=0}^{n} s_j 2^j, \; s_i \in \{0, 1, (-1)\}$$

Ein Vektor $(s_n s_{n-1} \cdots s_1 s_0)$*, für den gilt:* $s_i \cdot s_{i+1} = 0, \; 0 \leq i < n$ *sei definiert als eine NAF.*

Bei Verwendung von Faktoren in dieser Form wird ausgenutzt, dass die Negation eines Punktes auf einer elliptischen Kurve nur geringe Kosten verursacht. Bei der Berechnung einer NAF aus der Binärdarstellung einer ganzen Zahl werden Einserblöcke, die länger als 2 Bit sind, mit führender Null substituiert. Dabei wird die führende Null durch eine Eins, die letzte Eins durch eine minus Eins und die übrigen Einsen des Blocks durch Nullen ersetzt. In einer NAF existieren keine zwei aufeinander folgenden Werte, von denen nicht einer gleich Null ist.

$$\underbrace{1 \ldots 1}_{n-mal} = 1 \underbrace{0 \ldots 0(-1)}_{n-mal}, \; n \geq 2$$

Berechnen der NAF [5]:

Algorithmus 7 NAF-Berechnung

$\mathtt{Eingabe:}$ $\mathtt{Eine}\ l-\mathtt{Bit\ lange\ ganze\ Zahl}\ \mathtt{k} = \sum_{\mathtt{j=0}}^{\mathtt{l-1}} \mathtt{k_j 2^j}, \; \mathtt{k_j} \in \{0, 1\}$

$\mathtt{Ausgabe:}$ $\mathtt{NAF\ k} = \sum_{\mathtt{j=0}}^{\mathtt{l}} \mathtt{s_j 2^j}, \; \mathtt{s_j} \in \{-1, 0, 1\}$

$\mathtt{c_0} \leftarrow 0$
$\mathtt{For\ j = 0\ to\ l\ do:}$
$\quad \mathtt{c_{j+1}} \leftarrow \lfloor (\mathtt{k_j} + \mathtt{k_{j+1}} + \mathtt{c_j})/2 \rfloor$
$\quad \mathtt{s_j} \leftarrow \mathtt{k_j} + \mathtt{c_j} - 2\mathtt{c_{j+1}}$
$\mathtt{Return}\ (\mathtt{s_l s_{j-1} \ldots s_0})$

Die NAF einer ganzen Zahl ist eindeutig und die Anzahl der Werte ungleich Null ist minimal.

Satz 3.5 *Eine ganze Zahl k besitzt eine eindeutige NAF.*

Beweis durch Widerspruch [12]. Eine ganze Zahl k besitzt zwei NAFs.

$$k = \sum_{j=0}^{l} s_j 2^j = \sum_{j=0}^{l'} s'_j 2^j, \ s_j \in \{(-1), 0, 1\}$$

Sei j die kleinste Zahl mit $s_j \neq s'_j$. Es gelte o.B.d.A. $j = 0$.

k **gerade:** Wenn k gerade ist, muss $s_0 = 0 = s'_0$. Widerspruch zu der Voraussetzung, dass $s_j \neq s'_j$ ist.

k **ungerade:** Annahme: $c_0 = 1,\ c'_0 = (-1)$.
Da die beiden Darstellungen NAFs sind, muss $c_1 = c'_1 = 0$ gelten. Für die Summe $2k$ der beiden NAFs gilt: $c_1 = c'_1 = 0$ und $c_0 = c'_0 = 0$. Diese Zahl ist durch 4 teilbar. Daraus ergibt sich ein Widerspruch damit, dass k ungerade ist.

□

Satz 3.6 *Die Anzahl der Werte ungleich Null einer NAF sind minimal über alle Vektoren* $k = (k_n \cdots k_1 k_0),\ k \in \{(-1), 0, 1\}$.

Sei k in einer beliebigen Reihendarstellung

$$\sum_{j=0}^{l} s_j 2^j, \ s_j \in \{(-1), 0, 1\}$$

Zu zeigen ist, dass die Reihendarstellung in eine NAF transformiert werden kann, ohne dass dabei das Hamming-Gewicht erhöht wird [12]. Sei j der minimale Index, sodass $s_j \neq 0 \neq s_{j+1}$ gilt.
Zu unterscheiden sind folgende Fälle:

1. $s_j = (-1)$ und $s_{j+1} = 1$(analog zu $s_j = 1$ und $s_{j+1} = (-1)$)
2. $s_j = 1$ und $s_{j+1} = 1$(analog zu $s_j = (-1)$ und $s_{j+1} = (-1)$)

1. Fall: Die Bitfolge 1 (-1) wird in einer NAF durch 0 1 dargestellt. Das Hamming-Gewicht ist um Eins verringert.

2. Fall: Die Bitfolge 1 1 wird in einer NAF durch 1 0 (-1) dargestellt. In Abhängigkeit vom Wert s_{j+2} bleibt das Hamming-Gewicht erhalten oder es wird verringert.

Durch das wiederholte Anwenden der Operationen wird die NAF der Zahl k bestimmt, wobei sich das Hamming-Gewicht nicht erhöht. Daraus folgt, dass keinen Vektor $(s_l \cdots s_1 s_0),\ s_i \in \{(-1), 0, 1\}$ existiert, für den gilt:

$$k = \sum_{j=0}^{l} s_j 2^j, \ s_j \in \{(-1), 0, 1\}$$

der ein niedrigeres Hamming-Gewicht hat als eine NAF.

□

Die NAF kann einem leicht modifizierten Multiplizierer durch wiederholtes Verdoppeln als Faktor übergeben werden.

Algorithmus 8 NAF-Multiplizierer

```
Eingabe : P ∈ E_𝕂, k = Σ_{j=0}^{l−1} s_j 2^j, s_j ∈ {(−1), 0, 1}
Ausgabe : S = [k] P
S ← 𝒪
For i = 0 to l − 1 do :
    If k_i == 1 then S ← S + P.
    If k_i == (−1) then S ← S − P.
    P ← P + P
Return S
```

3.4.4 m-ärer Sliding-Window-Multiplizierer

m-äre Sliding-Window-Multiplizierer basieren auf dem Horner-Schema. Bei dieser Methode wird die Binärdarstellung des Faktors, mit dem ein Punkt multipliziert werden soll, in Blöcke, sogenannte Fenster, eingeteilt. Die maximale Größe eines solchen Fensters sei bezeichnet mit r. Die einzelnen Fenster müssen nicht aneinander anschließen. Schließt ein Block von Nullen an ein Fenster an, so beginnt das nächste Fenster bei der ersten folgenden Eins.

Beispiel:
Sei $k = 43526641 = (10100110000010100111110001)_2$ und $r = 4$.

$$\underline{101}\ 00\ \underline{11}\ 00000\ \underline{1}\ 0\ \underline{1001}\ \underline{1111}\ 000\ \underline{1}$$

Das Produkt $[k]\,P$ wird bestimmt durch:

$$[k]\,P = 2^{23}(5 \cdot P) + 2^{19}(3 \cdot P) + 2^{13}(1 \cdot P) + 2^{8}(9 \cdot P) + 2^{4}(15 \cdot P) + 2^{0}(1 \cdot P)$$

Die einzelnen Fenster werden so transformiert, dass sie nur Werte zwischen $-2^{r-2} + 1$ und $2^{r-2} - 1$ enthalten. Dabei ist zu beachten, dass der dem Fenster nachfolgende Bitvektor bei einen negativen Fensterwert um Eins inkrementiert werden muss.

Definition 3.9 $\overline{1}$ *sei definiert als* -1.

$$\underline{101}\ 00\ \underline{11}\ 00000\ \underline{1}\ 0\ \underline{1001}\ \underline{1111}\ 000\ \underline{1} =$$

$$\underline{101}\ 00\ \underline{11}\ 00000\ \underline{1}\ 0\ \underline{101}0\ 000\underline{\overline{1}}\ 000\underline{1}$$

$$2^{23}(5 \cdot P) + 2^{19}(3 \cdot P) + 2^{13}(1 \cdot P) + 2^{8}(9 \cdot P) + 2^{4}(15 \cdot P) + 2^{0}(1 \cdot P) =$$

$$2^{23}(5 \cdot P) + 2^{19}(3 \cdot P) + 2^{13}(1 \cdot P) + 2^{9}(5 \cdot P) + 2^{4}((-1) \cdot P) + 2^{0}(1 \cdot P)$$

Diese Darstellung wird als *Decomposition* bezeichnet. Repräsentativ für diese Darstellung müssen die Exponenten mit den dazugehörigen Faktoren gespeichert werden.

$$((1,0),(-1,4),(5,9),(1,13),(3,19),(5,23))$$

Die Decomposition wird mit folgendem Algorithmus berechnet [5]:

Algorithmus 9 Decomposition-Berechnung

$\mathtt{Eingabe:}$ $\mathtt{Eine}$ $\mathtt{l-Bit\ lange\ ganze\ Zahl}$ $\mathtt{k} = \sum_{\mathtt{j}=0}^{\mathtt{l}-1} \mathtt{k_j} 2^{\mathtt{j}},\ \mathtt{k_j} \in \{0,1\}$

$\mathtt{Ausgabe:}$ $\mathtt{Tupelvektor}\{(\mathtt{b_i},\mathtt{e_i})\}_{\mathtt{i}=0}^{\mathtt{d}-1}$

$\mathtt{d} := 0,\ \mathtt{j} := 0$
$\mathtt{While}\ \mathtt{j} \leq \mathtt{l}\ \mathtt{do:}$
    $\mathtt{If}\ \mathtt{k_j} = 0\ \mathtt{then}\ \mathtt{j} := \mathtt{j} + 1$
    $\mathtt{Else\ do:}$
      $\mathtt{t} := \mathtt{min}\{\mathtt{l}, \mathtt{j} + \mathtt{r} - 1\},\ \mathtt{h_d} := (\mathtt{k_t k_{t-1}} \ldots \mathtt{k_j})_2$
      $\mathtt{If}\ \mathtt{h_d} > 2^{\mathtt{r}-1}\ \mathtt{then\ do:}$
        $\mathtt{b_d} := \mathtt{h_d} - 2^{\mathtt{r}},$
        $\mathtt{erhöhe}\ (\mathtt{k_l k_{l-1}} \ldots \mathtt{k_{t+1}})_2\ \mathtt{um}\ 1.$
      $\mathtt{Else}\ \mathtt{b_d} := \mathtt{h_d}$
      $\mathtt{e_d} := \mathtt{j},\ \mathtt{d} := \mathtt{d} + 1,\ \mathtt{j} := \mathtt{t} + 1$
$\mathtt{Return\ den\ Vektor}\ (\mathtt{b_0},\mathtt{e_0}),(\mathtt{b_1},\mathtt{e_1}),\ldots,(\mathtt{b_{d-1}},\mathtt{e_{d-1}})$

Die wenigen Werte von $\left[-2^{r-2}-1\right]P$ bis $\left[2^{r-2}-1\right]P$ können vorberechnet und gespeichert werden, um den Multiplizierer zu optimieren. Dabei müssen real nur die Werte von $[2]\,P$ bis $\left[2^{r-2}-1\right]P$ berechnet werden.
Der folgende Algorithmus berechnet mit der Decomposition von k das Produkt von k und einem Punkt $P \in E_{\mathbb{K}}$ [5]:

Algorithmus 10 m-ärer Sliding-Window-Multiplizierer

$\mathtt{Eingabe:}\ \mathtt{P} \in \mathtt{E}_{\mathbb{K}}, \{(\mathtt{b_i}, \mathtt{e_i})\}_{\mathtt{i=0}}^{\mathtt{d-1}},\ \mathtt{sodass}\ \mathtt{k} = \sum_{\mathtt{i=0}}^{\mathtt{d-1}} \mathtt{b_i} 2^{\mathtt{e_j}}$

$\mathtt{Ausgabe:}\ \mathtt{S} = [\mathtt{k}]\,\mathtt{P}$

Vorberechnung

$\mathtt{P_1} := \mathtt{P},\ \mathtt{P_2} := [2]\,\mathtt{P}$

$\mathtt{For}\ \mathtt{i} = 1\ \mathtt{to}\ 2^{\mathtt{r}-2} - 1\ \mathtt{do:}$

$\quad \mathtt{P_{2i+1}} := \mathtt{P_{2i-1}} + \mathtt{P_2}$

$\mathtt{S} := \mathtt{P_{b_{d-1}}}$

Hauptprogramm

$\mathtt{For}\ \mathtt{i} = \mathtt{d} - 2\ \mathtt{to}\ \mathtt{by}\ -1\ \mathtt{do:}$

$\quad \mathtt{S} := [2^{\mathtt{e_{i+1}} - \mathtt{e_i}}]\,\mathtt{S}$

$\quad \mathtt{If}\ \mathtt{b_i} > 0\ \mathtt{then}\ \mathtt{S} := \mathtt{S} + \mathtt{P_{b_i}}$

$\quad \mathtt{Else}\ \mathtt{S} := \mathtt{S} - \mathtt{P_{-b_i}}$

$\mathtt{S} := [2^{\mathtt{e_0}}]\,\mathtt{S}$

$\mathtt{Return}\ \mathtt{S}$

4 Laufzeitanalyse

Die verschiedenen Algorithmen und Koordinatensysteme bieten unterschiedliche Vor- und Nachteile. Die Laufzeiten sind abhängig von den arithmetischen Operationen. Im Folgenden kennzeichnet M die modulare Multiplikation, S die modulare Quadrierung und A die modulare Addition und Multiplikation mit konstanten Faktoren. In den verschiedenen Algorithmen zur Punktaddition und Punktverdopplung ist der größte konstante Faktor zur arithmetischen Multiplikation 8; daher werden Multiplikationen mit konstanten Faktoren ≤ 8 nicht wie reguläre Multiplikationen betrachtet. Die allgemeine arithmetische Multiplikation würde für diese Operationen die Laufzeit zu stark beeinflussen. Diese Multiplikationen können durch Shifts und einzelne Additionen bestimmt werden. Das Produkt $3 \cdot a$ kann durch einen Linksshift von a, gleich $2 \cdot a$, und einer Addition mit a berechnet werden; $2 \cdot a + a = 3 \cdot a$. Im Algorithmus können Zwischenergebnisse und Exponentierungen eines festen Wertes mehrfach auftreten. Diese werden bei der Zählung der Operationen nur einmal aufgeführt. Soll a^3 berechnet werden, so wird $a^2 \cdot a$ berechnet. Wenn a^2 bereits bestimmt wurde muss nur eine Multiplikation berechnet werden. Werden Zwischenergebnisse wiederverwendet, werden diese bei der Analyse mit $*$ gekennzeichnet. Das Produkt zweier Elemente $a, b \in \mathbb{F}_p$ kann mit $O(n^2)$ Bitoperationen berechnet werden [7]. Es existieren Algorithmen, die das Quadrat einer ganzen Zahl $a \in \mathbb{F}_p$ etwas schneller berechnen als allgemeine Multiplikationsalgorithmen. Es wird hier jedoch angenommen, dass die Kosten für eine modulare Quadrierung $O(n^2)$ sind. Eine modulare Addition hat die Kosten $O(n)$. Die Kosten für *If*-Abfragen und die Kosten zur Erhöhung von Zählern in Schleifen werden vernachlässigt.

4.1 Laufzeitunterschiede zwischen den verschiedenen Koordinatensystemen

Im Folgenden werden die Additions- und Verdopplungsalgorithmen auf elliptischen Kurven in den verschiedenen Koordinatensystemen analysiert. Dabei ist zu beachten, dass die Punktaddition eines Punktes einer elliptischen Kurve mit dem neutralen Element der Addition keine Kosten verursacht, genauso wie die Verdopplung des neutralen Elementes der Addition. Im projektiven Raum und im Jacobi-Raum ist nicht immer ohne Berechnung eines multiplikativen Inversen zu erkennen, ob zwei Punkte gleich sind. In der Praxis werden immer zuerst zwei Punkte mit unterschiedlichen Koordinaten addiert. Tritt hierbei ein Fehler auf, müssen die Punkte gleich sein und einer der Punkte wird verdoppelt. Dies passiert bei skalaren Multiplikationen selten. Der Fall, das zwei Punkte addiert werden, obwohl sie gleich sind und anschliessend ein Punkt verdoppelt werden muss, wird in der Analyse nicht betrachtet.

4.1.1 Primkörper

In jedem Koordinatensystem verursacht die Addition eines Punktes P mit dem neutralen Element keine Kosten. Daher muss nur Rechenzeit für die Addition zweier verschiedener Punkte und die Verdopplung eines Punktes untersucht werden.

In der affinen Darstellung einer elliptischen Kurve über einem Primkörper ist das Inverse eines Punktes eindeutig. Soll ein Punkt P mit seinem Inversen $-P$ addiert werden, verursacht dies keine Kosten. Die Verdopplung eines Punktes P mit $y = 0$ verursacht keine Kosten, da dieser Punkt selbstinvers ist. In allen anderen Fällen muss für die Addition zweier Punkte (S. 10) und für die Verdopplung eines Punktes (S. 11) eine Inversion berechnet werden. Die Berechnung eines multiplikativen Inversen verursacht hohe Kosten im Vergleich zu Additionen und Multiplikationen. Das multiplikative Inverse eines Elementes $a \in \mathbb{F}_p$ kann mit dem erweiterten euklidischen Algorithmus mit $O(n^3)$ Bitoperationen berechnet werden [7]. Es gibt schnellere Algorithmen um eine Zahl zu invertieren, jedoch wird hier zur Analyse nur der erweiterte euklidische Algorithmus betrachtet.

	Addition
(1)	$1I + 1M + 2A$
(2)	$1S + 2A$
(3)	$1M + 2A$
Summe	$1I + 2M + 1S + 6A$

	Verdopplung
(4)	$1I + 1M + 1S + 4A$
(5)	$1S + 2A$
(6)	$1M + 2A$
Summe	$1I + 2M + 2S + 7A$

Definition 4.1 *Die affine Ebene sei bezeichnet durch $\mathbb{A}^2$, der projektive Raum durch $\mathbb{P}^3$ und der Jacobi-Raum durch $\mathbb{J}^3$.*

Um einen Punkt $P \in E_{\mathbb{F}_p}$ aus der affinen Ebene in den projektiven Raum abzubilden, werden die zweidimensionalen Koordinaten um die Z-Koordinate erweitert und diese gleich 1 gesetzt.

$$i : \mathbb{A}^2(E_{\mathbb{F}_p}) \to \mathbb{P}^3(E_{\mathbb{F}_p}),$$
$$i : (x, y) \to (x, y, 1)$$

Bei den Punktoperationen im projektiven Raum muss kein multiplikatives Inverses berechnet werden. Dafür werden bei Addition und Verdopplung (S. 12) mehr Berechnungen benötigt. Ein weiterer Nachteil bei der Verwendung elliptischer Kurven im projektiven Raum ist, dass ein Punkt mehrere Darstellungen hat, da $P = (X, Y, Z) = (tX, tY, tZ) = P'$ für jedes $t \in \mathbb{F}_p^*$. Um zu bestimmen, ob zwei Punkte gleich sind, muss ein multiplikatives Inverses oder eine vollständige Punktaddition berechnet werden. Um einen Punkt in projektiver Koordinatendarstellung wieder in die affine Ebene zu projizieren, muss das multiplikative Inverse der Z-Koordinate berechnet werden.

$$j : \mathbb{P}^3(E_{\mathbb{F}_p}) \rightarrow \mathbb{A}^2(E_{\mathbb{F}_p})$$
$$j : (X, Y, Z) \rightarrow (XZ^{-1}, YZ^{-1})$$

	Addition
(7)	$2M + 1A$
(8)	$2M + 1A$
(9)	$5M + 2S + 3A\ *$
(10)	$1M$
(11)	$3M + 2A\ *$
(12)	$2M\ *$
Summe	$15M + 2S + 7A$

	Verdopplung
(13)	$1M + 2S + 2A$
(14)	$1M$
(15)	$2M$
(16)	$1S + 2A$
(17)	$1M + 1A$
(18)	$2M + 2S + 4A$
(19)	$1M + 1A\ *$
Summe	$8M + 5S + 10A$

Elliptische Kurven im Jacobi-Raum haben die gleichen Vor- und Nachteile wie elliptische Kurven im projektiven Raum. Kurvenoperationen im Jacobi-Raum (S. 13) benötigen weniger arithmetische Operationen als im projektiven Raum. Um einen Punkt aus dem Jacobi-Raum in die affine Ebene zu projizieren, muss das multiplikative Inverse der Z-Koordinate berechnet werden. Es gilt:

$$k : \mathbb{J}^3(E_{\mathbb{F}_p}) \rightarrow \mathbb{A}^2(E_{\mathbb{F}_p})$$

$$k : (X, Y, Z) \rightarrow (XZ^{-2}, YZ^{-3})$$

	Addition
(20)	$1M + 1S$
(21)	$1M + 1S$
(22)	$2M\ *$
(23)	$2M\ *$
(24)	$1A$
(25)	$1A$
(26)	$2M + 2S + 3A\ *$
(27)	$2M + 2A\ *$
(28)	$2M$
Summe	$14M + 4S + 6A$

	Verdopplung
(29)	$1M + 1S + 1A$
(30)	$1M + 3S + 2A$
(31)	$1S + 2A$
(32)	$-$
(33)	$1M + 1S + 3A$
(34)	$1M + 1A$
Summe	$4M + 6S + 9A$

Die Gesamtzahl der arithmetischen Operationen gibt Aufschluss über die Einsetzbarkeit der verschiedenen Koordinatensysteme. Punktadditionen auf elliptischen Kurven in affiner Darstellung benötigen nur wenige Multiplikationen und Additionen, jedoch die Berechnung eines multiplikativen Inversen. Bei jeder Punktaddition und Punktverdopplung muss ein anderes multiplikatives Inverses berechnet werden. Bei Punktadditionen und Punktverdopplungen auf elliptischen Kurven in projektiver Darstellung und in Jacobidarstellung muss kein multiplikatives Inverses berechnet werden. Solche Kurvenoperationen benötigen jedoch mehr Additionen und Multiplikationen als Kurvenoperationen in der affinen Ebene.

	affin	projektiv	Jacobi
Addition	$1I + 2M + 1S + 6A$	$15M + 2S + 7A$	$14M + 4S + 6A$
Verdopplung	$1I + 2M + 2S + 7A$	$8M + 5S + 10A$	$4M + 6S + 9A$

Eine detaillierte Analyse wird in 4.3 durchgeführt.

4.1.2 Körper der Charakteristik 2

Die Laufzeiten der verschiedenen arithmetischen Operationen in Körpern der Charakteristik 2 sind im O-Kalkül die gleichen, wie die Laufzeiten in Primkörpern. Zur Optimierung der Kurvenoperationen werden in der Praxis Ergebnisse wiederverwendet, genauso wie bei Kurvenoperationen auf elliptischen Kurven über Primkörpern.

Definition 4.2 *Sei a ein Element eines Polynomkörpers der Charakteristik 2 $a \in \mathbb{F}_{2^q}$. Der Koeffizientenvektor von a sei definiert als Vektor der Koeffizienten von a.*

$$a_{n-1} \cdot x^{n-1} + a_{n-2} \cdot x^{n-2} + \cdots + a_1 \cdot x + a_0 = (a_{n-1}a_{n-2} \cdots a_1 a_0),\ a_i \in \{0, 1\}$$

Bei Verwendung elliptischer Kurven über Körpern der Charakteristik 2 in der affinen Ebene muss bei der Punktaddition (S. 13) und bei der Punktverdopplung (S. 14) ein kostenintensives multiplikatives Inverses berechnet werden. Die Anzahl der Multiplikationen, Quadrierungen und Additionen ist jedoch gering.

	Addition
(35)	$1I + 1M + 2A$
(36)	$1S + 4A$
(37)	$1M + 3A$
Summe	$1I + 2M + 1S + 9A$

	Verdopplung
(38)	$1I + 1M + 1A$
(39)	$1S + 2A$
(40)	$1M + 3A$
Summe	$1I + 2M + 1S + 6A$

Um einen Punkt $P \in E_{\mathbb{F}_{2^q}}$ aus der affinen Ebene in den projektiven Raum abzubilden, werden die zweidimensionalen Koordinaten um die Z-Koordinate erweitert und diese gleich 1 gesetzt.

$$i : \mathbb{A}^2(E_{\mathbb{F}_{2^q}}) \to \mathbb{P}^3(E_{\mathbb{F}_{2^q}}),$$

$$i : (x, y) \to (x, y, 1)$$

Die Addition zweier Punkte $P, Q \in \mathbb{F}_{2^q}$ (S. 14) und die Verdopplung (S. 14) eines Punktes $P \in \mathbb{F}_{2^q}$ benötigt im projektiven Raum folgende Operationen:

	Addition
(41)	$1M + 1S$
(42)	$1M + 1S$
(43)	$1A$
(44)	$2M *$
(45)	$2M *$
(46)	$1A$
(47)	$1M$
(48)	$2M + 1A$
(49)	$1M$
(50)	$1A$
(51)	$3M + 2S + 2A$
(52)	$2M + 1S + 1A$
Summe	$15M + 5S + 7A$

	Verdopplung
(53)	$1M + 1S$
(54)	$1M + 2S + 1A *$
(55)	$1M + 1S + 2A$
(56)	$2M + 1S + 1A *$
Summe	$5M + 5S + 4A$

Um einen Punkt aus dem projektiven Raum in die affine Ebene abzubilden, muss das multiplikative Inverse der Z-Koordinate eines Punktes $P = (X, Y, Z)$ berechnet werden. Es gilt:

$$j : \mathbb{P}^3(E_{\mathbb{F}_{2^q}}) \rightarrow \mathbb{A}^2(E_{\mathbb{F}_{2^q}})$$

$$j : (X, Y, Z) \rightarrow (XZ^{-2}, YZ^{-3})$$

	affin	projektiv
Addition	$1I + 2M + 1S + 9A$	$15M + 5S + 7A$
Verdopplung	$1I + 2M + 1S + 6A$	$5M + 5S + 4A$

Eine detaillierte Analyse wird in 4.3 durchgeführt.

4.2 Laufzeitunterschiede zwischen den verschiedenen Reduktionsalgorithmen

Für modulare Reduktionen mittels Barrett- und Montgomery-Algorithmus muss eine Basis b gewählt werden. Bei dem Montgomery-Algorithmus muss b teilerfremd zum Körper sein. Wenn in der ECC endliche Primkörper verwendet werden, kann 2 als Basis b für die Montgomery-Reduktion und Barrett-Reduktion gewählt werden.

Definition 4.3 *Sei a eine beliebige ganze Zahl, dann bezeichnet $\#a$ die Anzahl der Bits der Binärdarstellung von a.*
Sei b ein Polynom aus einem Polynomkörper der Charakteristik 2, dann bezeichnet $\#b$ die Länge des Koeffizientenvektors von b.

Die folgenden Algorithmen bestimmen die modulare Reduktion $z \equiv x \pmod{p}$, $0 \leq z < p < x$.
$x \mod p$ in einem Körper $\mathbb{F}_p$ kann mittels ganzzahliger Division, Multiplikation und Subtraktion berechnet werden. Dieses Verfahren wird als Standardreduktion bezeichnet. Die ganzzahlige Division $\lfloor x/p \rfloor$, $x > p$ kann mit maximal $(\#x - \#p + 1) \cdot \#p$ Bitoperationen berechnet werden.

Beispiel 1: $(1111101)_2/(1101)_2$
1111101/1101 = 1001
<u>1101</u>
101
<u>0</u>
1010
<u>0</u>
10101
<u>1101</u>
1000

Beispiel 2: $(1111)_2/(1101)_2$
1111/1101 = 1
<u>1101</u>
10

Bei der Division $\lfloor x/p \rfloor$ müssen maximal $(\#x - \#p + 1)$ Subtraktionen berechnet werden. Eine Subtraktion benötigt $\#p$ Bitoperationen.

Operation	Laufzeit	Ausgabelänge
$\mathtt{q} := \lfloor \mathtt{x}/\mathtt{p} \rfloor$	$(\#x - \#p + 1) \cdot \#p$	$\#x - \#p + 1$
$\mathtt{r} := \mathtt{q} \cdot \mathtt{p}$	$(\#x - \#p + 1) \cdot \#p$	$\#x$
$\mathtt{z} := \mathtt{x} - \mathtt{r}$	$\#x$	$\#p$
`Return z`	1	$\#p$
Summe	$2 \cdot (\#x \cdot \#p - \#p^2 + \#p) + \#x + 1$	

4.2.1 Analyse der Barrett-Reduktion

Der Einsatz der Barrett-Reduktion ist nur dann sinnvoll, wenn viele modulare Reduktionen in einem festen Körper berechnet werden müssen. Daher müssen die Kosten, die bei der Berechnung von $\mu = \lfloor b^{2t}/p \rfloor$ entstehen nicht beachtet werden. In Hardware können Multiplikationen mit festen Zweierpotenzen (Abbildung 7) und Divisionen durch Zweierpotenzen (Abbildung 8) durch Shifts in einer Bitoperation berechnet werden. Dies ist möglich, da durch die Parallelität alle Bits auf einmal verschoben werden können. Bei der ganzzahligen Division durch 2^n werden die niederwertigsten n Bits gestrichen. Die Bitlänge der Konstanten μ sei als $\#\mu \leq \#p$ bezeichnet. Die Reduktion $x \pmod{2^k}$, $x = x_{k+l} \cdots x_k x_{k-1} \cdots x_1 x_0$, $k, l \in \mathbb{N}$ kann mit einer Bitoperation berechnet werden (Abbildung 9).

$$x_{k-1} \cdots x_1 x_0 \equiv x \pmod{2^k}$$

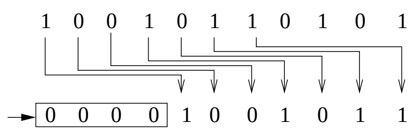

Abbildung 7:
$(10101101001)_2/2^4$

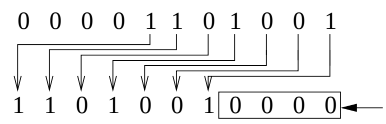

Abbildung 8:
$(1101001)_2 \cdot 2^4$

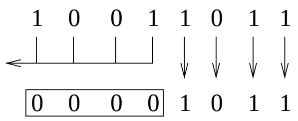

Abbildung 9: $(10011011)_2 \mod 2^4$

Operation	Laufzeit	Ausgabelänge
$\mathtt{q_0} := \lfloor \mathtt{x}/\mathtt{b}^{\mathtt{t}-1} \rfloor$	1	$\#x - \#p$
$\mathtt{q} := \lfloor (\mu \mathtt{q_0})/\mathtt{b}^{\mathtt{t}+1} \rfloor$	$(\#x - \#p) \cdot \#\mu + 2$	$\#x - (\#p + 1)$
$\mathtt{r_1} := \mathtt{x} \pmod{\mathtt{b}^{\mathtt{t}+1}}$	1	$\#p + 1$
$\mathtt{r_2} := \mathtt{qp} \pmod{\mathtt{b}^{\mathtt{t}+1}}$	$(\#x - \#p) \cdot \#p + 1$	$\#p + 1$
$\mathtt{r} := \mathtt{r_2} - \mathtt{r_1}$	$\#p + 1$	$\#p + 1$
$\mathtt{If\ z} < 0\ \mathtt{then\ z} := \mathtt{z} + \mathtt{b}^{\mathtt{t}+1}$	$\#p + 1$ *(optional)*	$\#p$
$\mathtt{While\ z} \geq \mathtt{p\ do\ z} := \mathtt{z} - \mathtt{p}$	$2 \cdot \#p$ *(optional)*	$\#p$
`Return z`	1	$\#p$
Summe	$(\#x - \#p + 4) \cdot \#p$ $+(\#x - \#p) \cdot \#\mu + 8$	

4.2.2 Analyse der Montgomery-Reduktion

Der Einsatz der Montgomery-Reduktion ist nur bei vielen modularen Reduktionen in einem festen Körper zu empfehlen. Die Kosten für die Berechnung von $p' = -p^{-1} \pmod{R}$ werden nicht beachtet, da dies nur einmal berechnet werden muss und für alle Reduktionen im Körper gleich ist.

Operation	Laufzeit	Ausgabelänge
$\mathtt{For\ i} = 0\ \mathtt{to\ n} - 1\ \mathtt{do:}$	$\#p - mal$	
$\quad \mathtt{u} := \mathtt{y_i p'} \pmod{\mathtt{b}}$	1	1
$\quad \mathtt{y} := \mathtt{y} + \mathtt{upb^i}$	$\#p$	$2 \cdot \#p + 1$
$\mathtt{z} := \mathtt{y}/\mathtt{R}$	1	$\#p + 1$
$\mathtt{If\ z} \geq \mathtt{p\ then\ z} := \mathtt{z} - \mathtt{p}$	$\#p$ *(optional)*	$\#p$
`Return z`	1	$\#p$
Summe	$2 \cdot \#p^2 + 2 \cdot \#p + 2$	

Die Montgomery-Reduktion verursacht die geringsten Kosten, jedoch ist das Ergebnis in Montgomerydarstellung und es muss wieder in die Normaldarstellung umgerechnet werden. Die Barrett-Reduktion ist kostenintensiver als die Montgomery-Reduktion, aber kostengünstiger als die Standardreduktion.
Nach der Analyse ist die Montgomery-Reduktion das kostengünstigste Verfahren zur modularen Reduktion. Dieser Algorithmus empfiehlt sich, wenn nur wenige Zwischenergebnisse reduziert werden müssen. Bei Exponentierungsverfahren wie *repeated-squaring* ist die Montgomery-Reduktion eine gute Wahl, da immer nur ein unabhängiger Wert reduziert werden muss. Bei Punktadditionen und Punktverdopplungen müssen viele Zwischenergebnisse berechnet werden, wobei ständig zwischen Montgomery- und Normaldarstellung gewechselt werden muss. Für den Wechsel zwischen den beiden Darstellungen kann die Montgomery-Multiplikation verwendet werden; dadurch steigt jedoch die benötigte Rechenzeit so stark an, dass die Vorteile der Montgomery-Reduktion gegenüber der

Barrett-Reduktion verfallen. Daher ist der Einsatz der Barrett-Reduktion zu empfehlen, da dieses Verfahren der kostengünstigste Algorithmus ist.

4.3 Laufzeitunterschiede zwischen Punktverdopplung und Punktaddition

Bei *Zeitangriffen* und der *Differential Power Analysis* auf kryptographische Systeme kann der Laufzeitunterschied zwischen den verschiedenen arithmetischen Operationen genutzt werden, um den geheimen Schlüssel zu bestimmen [16]. Bei Systemen, die auf elliptischen Kurven basieren kann zudem der Unterschied zwischen Addition und Verdopplung ausgenutzt werden. Eine Untersuchung der Laufzeitunterschiede in den verschiedenen Koordinatensystemen liefert somit Informationen über die Angreifbarkeit der unterschiedlichen Implementierungen. Bei der modularen Reduktion werden bei der Laufzeitanalyse zwei Fälle unterschieden. Die Ergebnisse allgemeiner Multiplikationen und Quadrierungen werden mit einem der in Abschnitt 3.2 analysierten Algorithmen reduziert; hierfür wird die Laufzeit $O(n^2)$ angenommen. Diese Reduktionen werden für die Analyse mit A gekennzeichnet. Additionen und Multiplikationen mit Konstanten ≤ 8 werden durch Addition bestimmt. Hierfür wird die Laufzeit $O(n)$ angenommen. Diese Ergebnisse können durch Subtraktion reduziert werden. Dabei muss von einer ganzen Zahl x, für die gilt $0 \leq x < 8p$ im worst case $7 \cdot p$ subtrahiert werden. Diese Reduktionen werden für die Analyse mit B gekennzeichnet und haben die Laufzeit $O(n)$. Um die Algorithmen allgemein zu vergleichen wird das Produkt zweier Zahlen nicht mittels Montgomery-Multiplikation bestimmt. Inversionen benötigen keine modularen Reduktionen. Die binäre Länge von p sei definiert als $\#p$. Dies ist die obere Schranke der binären Länge eines Wertes aus $\mathbb{F}_p$. Für die Analyse wird angenommen, dass die Berechnungen für die Punktaddition und Punkverdopplung optimiert sind; d.h. Zwischenergebnisse werden wiederverwendet und nur einmal berechnet.

4.3.1 Kurven in affiner Darstellung über $\mathbb{F}_p$

Die Anzahl der Operationen zwischen Punktaddition und Punktverdopplung in der affinen Ebene unterscheidet sich nur gering.

Addition	Bitoperationen	Anzahl der Reduktionen
(1)	$\#p + \#p + \#p^3 + \#p^2$	$A + 2 \cdot B$
(2)	$\#p^2 + \#p + \#p$	$A + 2 \cdot B$
(3)	$\#p + \#p^2 + \#p$	$A + 2 \cdot B$
Summe	$\#p^3 + 3 \cdot \#p^2 + 6 \cdot \#p$	$3 \cdot A + 6 \cdot B$

Bei der Punktaddition von zwei Punkten auf einer elliptischen Kurve über einem Primkörper in affiner Darstellung müssen im worst case 3 modulare Reduktionen mit einem Reduktionsalgorithmus und 6 modulare Reduktionen mittels Subtraktion berechnet werden.

Verdopplung	Bitoperationen	Anzahl der Reduktionen
(4)	$\#p^2 + 1 + \#p + \#p + 1 + \#p^3 + \#p^2$	$2 \cdot A + 3 \cdot B$
(5)	$\#p^2 + 1 + \#p$	$A + 2 \cdot B$
(6)	$\#p + \#p^2 + \#p$	$A + 2 \cdot B$
Summe	$\#p^3 + 4 \cdot \#p^2 + 5 \cdot \#p + 3$	$4 \cdot A + 7 \cdot B$

Bei der Verdopplung eines Punktes auf einer elliptischen Kurve über $\mathbb{F}_p$ in affiner Darstellung müssen im worst case 4 modulare Reduktionen mit einem Reduktionsalgorithmus und 7 modulare Reduktionen mittels Subtraktion berechnet werden. Die Verdopplung eines Punktes verursacht somit mehr Kosten als die Addition von zwei Punkten auf einer elliptischen Kurve in affiner Darstellung.

4.3.2 Kurven in projektiver Darstellung über $\mathbb{F}_p$

Bei der Punktaddition und Punktverdopplung im projektiven Raum muss keine Inversion berechnet werden. Operationen auf elliptischen Kurven im projektiven Raum benötigen mehr arithmetische Berechnungen als in der affinen Ebene. Für die Untersuchung, welche Kosten die Berechnung eines multiplikativen Inversen verursachen darf um in der affinen Ebene statt im projektiven Raum zu rechnen, werden hier die Punktadditions- und Punktverdopplungskosten aufgeführt. Für die verschiedenen Reduktionsalgorithmen, die in 4.2 analysiert wurden, sei die Laufzeit beschränkt durch $c_1 \cdot \#p^2$. Für Reduktionen, die mittels Subtraktion berechnet werden sei die Laufzeit beschränkt durch $c_2 \cdot \#p$.

Addition	Bitoperationen	Anzahl der Reduktionen
(7)	$\#p^2 + \#p^2 + \#p$	$2 \cdot A + B$
(8)	$\#p^2 + \#p^2 + \#p$	$2 \cdot A + B$
(9)	$(4 \cdot \#p^2 + \#p) + (3 \cdot \#p^2 + 1) + \#p$	$7 \cdot A + 3 \cdot B$
(10)	$\#p^2$	A
(11)	$(\#p + \#p^2) + (2 \cdot \#p^2) + \#p$	$3 \cdot A + 2 \cdot B$
(12)	$2 \cdot \#p^2$	$2 \cdot A$
Summe	$17 \cdot \#p^2 + 6 \cdot \#p + 1$	$17 \cdot A + 7 \cdot B$

Eine Addition zweier Punkte im projektiven Raum benötigt im worst case $17 \cdot \#p^2 + 6 \cdot \#p + 1$ Bitoperationen für die arithmetischen Berechnungen. Zusätzlich müssen 17 modulare Reduktionen mit einem Reduktionsalgorithmus und 7 modulare Reduktionen mit Subtraktionen berechnet werden.

Satz 4.1 *Die Berechnung eines multiplikativen Inversen darf soviel kosten wie 28 Multiplikationen, damit eine Punktaddition in der affinen Ebene günstiger ist als im projektiven Raum.*

Beweis: Aus den Ergebnissen der Analyse der Punktaddition in der affinen Ebene und im projektiven Raum ergibt sich folgendes Gleichungssystem:

$$x + 3 \cdot \#p^2 + 6 \cdot \#p + 3 \cdot c_1 \cdot \#p^2 + 6 \cdot c_2 \cdot \#p = 17 \cdot \#p^2 + 6 \cdot \#p + 1 + 17 \cdot c_1 \cdot \#p^2 + 7 \cdot c_2 \cdot \#p$$

$$\Leftrightarrow x = 14 \cdot \#p^2 + 1 + 14 \cdot c_1 \cdot \#p^2 + c_2 \cdot \#p$$

Da bei jedem Reduktionsalgorithmus $c_1 \geq 1$ gilt, kann abgeschätzt werden:

$$x \leq 28 \cdot \#p^2 \cdot c_1 + 1 + c_2 \cdot \#p$$

Da für eine Multiplikation $\#p^2$ Bitoperationen im worst case angenommen werden, ist die Punktaddition in der affinen Ebene sicher günstiger, wenn die Berechnung eines multiplikativen Inversen nicht teurer ist als 28 Multiplikationen. $\square$

Wird die Barrett-Reduktion oder die Montgomery-Reduktion verwendet können die Kosten genauer abgeschätzt werden.

Satz 4.2 *Wird die Barrett-Reduktion verwendet, darf das Berechnen eines multiplikativen Inversen nicht mehr kosten als* 42 *Multiplikationen und* 71 *Additionen, damit die Punktaddition in der affinen Ebene günstiger ist als im projektiven Raum.*

Beweis: Für die Reduktionen von Typ A werden die Bitoperationen $(\#y - \#p+4)\cdot\#p+(\#y-\#p)\cdot\#\mu+8$ eingesetzt; hierbei bezeichnet $\#y$ die Bitlänge der zu reduzierenden Zahl. Bei den Reduktionen von Typ B wird hier für die Konstante c_2 der Wert Eins eingesetzt. Daraus ergibt sich:

$$x = 14\cdot\#p^2+1+14\cdot((\#y-\#p+4)\cdot\#p+(\#y-\#p)\cdot\#\mu+8)+\#p$$

Es kann $\#y$ für eine worst case Analyse durch $2\cdot\#p$ substituiert werden.

$$\begin{aligned}
x &= 14\cdot\#p^2+1+14\cdot((2\cdot\#p-\#p+4)\cdot\#p+(2\cdot\#p-\#p)\cdot\#\mu+8)\\
&\quad+\#p\\
&\leq 14\cdot\#p^2+1+14\cdot((\#p+4)\cdot\#p+\#p\cdot\#\mu+8)+\#p\\
&= 14\cdot\#p^2+1+14\cdot(\#p^2+4\cdot\#p+\#p\cdot\#\mu+8)+\#p
\end{aligned}$$

$\#\mu$ kann durch $\#p+1$ substituiert werden.

$$\begin{aligned}
&\leq 14\cdot\#p^2+1+14\cdot(\#p^2+4\cdot\#p+\#p\cdot(\#p+1)+8)+\#p\\
&= 14\cdot\#p^2+1+14\cdot(\#p^2+4\cdot\#p+\#p^2+\#p+8)+\#p\\
&= 14\cdot\#p^2+1+14\cdot(2\cdot\#p^2+5\cdot\#p+8)+\#p\\
&= 14\cdot\#p^2+113+28\cdot\#p^2+71\cdot\#p\\
&= 42\cdot\#p^2+71\cdot\#p+113
\end{aligned}$$

$\square$

Satz 4.3 *Wird die Montgomery-Reduktion verwendet, darf das Berechnen eines multiplikativen Inversen nicht mehr kosten als* 42 *Multiplikationen und* 29 *Additionen, damit die Punktaddition in der affinen Ebene günstiger ist als im projektiven Raum.*

Beweis: Bei Verwendung der Montgomery-Reduktion können die Reduktionen von Typ A durch $2\cdot\#p^2+2\cdot\#p+2$ substituiert werden.

$$\begin{aligned}
x &= 14\cdot\#p^2+1+14\cdot(2\cdot\#p^2+2\cdot\#p+2)+\#p\\
&= 14\cdot\#p^2+1+28\cdot\#p^2+28\cdot\#p+28+\#p\\
&= 42\cdot\#p^2+29\cdot\#p+29
\end{aligned}$$

$\square$

Verdopplung	Bitoperationen	Anzahl der Reduktionen
(13)	$(2 \cdot \#p^2) + (\#p^2 + 1 + \#p) + \#p$	$3 \cdot A + 2 \cdot B$
(14)	$\#p^2$	A
(15)	$2 \cdot \#p^2$	$2 \cdot A$
(16)	$\#p^2 + \#p + 1$	$A + 2 \cdot B$
(17)	$\#p^2 + 1$	$A + B$
(18)	$(\#p^2 + \#p + 1) + (3 \cdot \#p^2 + 1) + \#p$	$4 \cdot A + 4 \cdot B$
(19)	$\#p^2 + 1$	$A + B$
Summe	$13 \cdot \#p^2 + 5 \cdot \#p + 6$	$13 \cdot A + 10 \cdot B$

Die Punktverdopplung benötigt für die arithmetischen Operationen im worst case $13 \cdot \#p^2 + 5 \cdot \#p + 6$ Bitoperationen. Zusätzlich werden 13 modulare Reduktionen mittels Reduktionsalgorithmus und 6 mittels Subtraktion berechnet.

Satz 4.4 *Die Berechnung eines multiplikativen Inversen darf soviel kosten wie 19 Multiplikationen, damit eine Punktverdopplung in der affinen Ebene günstiger ist als im projektiven Raum.*

Beweis: Aus den Ergebnissen der Analyse der Punktverdopplung in der affinen Ebene und im projektiven Raum ergibt sich folgendes Gleichungssystem:

$$x+3\cdot\#p^2+5\cdot\#p+3+4\cdot c_1\cdot\#p^2+7\cdot c_2\cdot\#p = 13\cdot\#p^2+5\cdot\#p+6+13\cdot c_1\cdot\#p^2+10\cdot c_2\cdot\#p$$

$$\Leftrightarrow x = 10 \cdot \#p^2 + 3 + 9 \cdot c_1 \cdot \#p^2 + 3 \cdot c_2 \cdot \#p$$

Da bei jedem Reduktionsalgorithmus $c_1 \geq 1$ gilt, kann abgeschätzt werden:

$$\leq 19 \cdot \#p^2 \cdot c_1 + 3 + 3 \cdot c_2 \cdot \#p$$

Daraus folgt, dass die Punktverdopplung in der affinen Ebene sicher günstiger ist , wenn die Berechnung eines multiplikativen Inversen nicht teurer ist als 19 Multiplikationen.

□

Werden die Barrett-Reduktion oder die Montgomery-Reduktion verwendet, können die oberen Schranken für die Kosten einer Inversion genauer abgeschätzt werden.

Satz 4.5 *Bei Verwendung der Barrett-Reduktion darf die Berechnung eines multiplikativen Inversen keine höheren Kosten verursachen als* 28 *Multiplikationen und* 48 *Additionen, damit eine Punktverdopplung in der affinen Ebene günstiger ist als im projektiven Raum.*

Beweis: Die Reduktionen vom Typ A werden durch $2 \cdot \#p^2 + 5 \cdot \#p + 8 \geq (\#y - \#p + 4) \cdot \#p + (\#y - \#p) \cdot \#\mu + 8$ substituiert.

$$\begin{aligned} x &= 10 \cdot \#p^2 + 3 + 9 \cdot (2 \cdot \#p^2 + 5 \cdot \#p + 8) + 3 \cdot \#p \\ &= 10 \cdot \#p^2 + 3 + (18 \cdot \#p^2 + 45 \cdot \#p + 72) + 3 \cdot \#p \\ &= 28 \cdot \#p^2 + 48 \cdot \#p + 75 \end{aligned}$$

□

Satz 4.6 *Bei Verwendung der Montgomery-Reduktion darf die Berechnung eines multiplikativen Inversen keine höheren Kosten verursachen als 28 Multiplikationen und 21 Additionen, damit eine Punktverdopplung in der affinen Ebene günstiger ist als im projektiven Raum.*

Beweis: Bei Verwendung der Montgomery-Reduktion können die Reduktionen von Typ A durch $2 \cdot \#p^2 + 2 \cdot \#p + 2$ substituiert werden.

$$\begin{aligned} x &= 10 \cdot \#p^2 + 3 + 9 \cdot (2 \cdot \#p^2 + 2 \cdot \#p + 2) + 3 \cdot \#p \\ &= 10 \cdot \#p^2 + 3 + (18 \cdot \#p^2 + 18 \cdot \#p + 18) + 3 \cdot \#p \\ &= 28 \cdot \#p^2 + 21 \cdot \#p + 21 \end{aligned}$$

□

4.3.3 Kurven in Jacobidarstellung über $\mathbb{F}_p$

Bei Punktadditionen und Punktverdopplungen im Jacobi-Raum muss kein multiplikatives Inverses berechnet werden. Die Punktaddition und die Punktverdopplung auf der elliptischen Kurve im Jacobi-Raum benötigt mehr Additionen und Multiplikationen als in der affinen Ebene, jedoch weniger als im projektiven Raum.

Addition	Bitoperationen	Anzahl der Reduktionen
(20)	$2 \cdot \#p^2$	$2 \cdot A$
(21)	$2 \cdot \#p^2$	$2 \cdot A$
(22)	$2 \cdot \#p^2$	$2 \cdot A$
(23)	$2 \cdot \#p^2$	$2 \cdot A$
(24)	$\#p$	B
(25)	$\#p$	B
(26)	$(2 \cdot \#p^2 + \#p) + (2 \cdot \#p^2 + 1) + \#p$	$4 \cdot A + 3 \cdot B$
(27)	$(\#p + \#p^2) + \#p^2 + \#p$	$2 \cdot A + 2 \cdot B$
(28)	$2 \cdot \#p^2$	$2 \cdot A$
Summe	$16 \cdot \#p^2 + 6 \cdot \#p + 1$	$18 \cdot A + 7 \cdot B$

Die Punktaddition auf elliptischen Kurven über Primkörpern im Jacobi-Raum benötigt für die arithmetischen Operationen im worst case $16 \cdot \#p^2 + 6 \cdot \#p + 1$ Bitoperationen. Zusätzlich müssen 18 modulare Reduktionen mittels eines der in 3.1 vorgestellten Reduktionsalgorithmen und 7 modulare Reduktionen mittels Subtraktion berechnet werden.

Satz 4.7 *Die Berechnung eines multiplikativen Inversen darf soviel kosten wie 28 Multiplikationen, damit eine Punktaddition in der affinen Ebene günstiger ist als im Jacobi-Raum.*

Beweis: Aus den Ergebnissen der Analyse der Punktaddition in der affinen Ebene und im Jacobi-Raum ergibt sich folgendes Gleichungssystem:

$$x+3\cdot\#p^2+6\cdot\#p+3\cdot c_1\cdot\#p^2+6\cdot c_2\cdot\#p = 16\cdot\#p^2+6\cdot\#p+1+18\cdot c_1\cdot\#p^2+7\cdot c_2\cdot\#p$$

$$\Leftrightarrow x = 13 \cdot \#p^2 + 1 + 15 \cdot c_1 \cdot \#p^2 + c_2 \cdot \#p$$

$$\leq 28 \cdot c_1 \cdot \#p^2 + c_2 \cdot \#p + 1$$

Da $c_1 > 1$ folgt, dass eine Punktaddition in der affinen Ebene günstiger als im Jacobi-Raum ist, wenn das Bestimmen eines multiplikativen weniger als 28 Multiplikationen kostet.

□

Bei der Verwendung von der Barrett-Reduktion oder der Montgomery-Reduktion kann die Abschätzung genauer gemacht werden.

Satz 4.8 *Eine Punktaddition in der affinen Ebene ist günstiger bei Verwendung der Barrett-Reduktion als im Jacobi-Raum, wenn die Bestimmung eines multiplikativen Inversen nicht kostenintensiver ist als* 43 *Multiplikationen und* 76 *Additionen.*

Beweis: Die Reduktionen vom Typ A werden durch $2 \cdot \#p^2 + 5 \cdot \#p + 8$ substituiert. Die Konstante c_2 wird gleich Eins gesetzt.

$$\begin{aligned} x &= 13 \cdot \#p^2 + 1 + 15 \cdot (2 \cdot \#p^2 + 5 \cdot \#p + 8) + \#p \\ &= 13 \cdot \#p^2 + 1 + (30 \cdot \#p^2 + 75 \cdot \#p + 120) + \#p \\ &= 43 \cdot \#p^2 + 76 \cdot \#p + 121 \end{aligned}$$

□

Satz 4.9 *Eine Punktaddition in der affinen Ebene ist günstiger bei Verwendung der Montgomery-Reduktion als im Jacobi-Raum, wenn die Bestimmung eines multiplikativen Inversen nicht kostenintensiver ist als* 43 *Multiplikationen und* 31 *Additionen.*

Beweis: Die Reduktionen vom Typ A werden durch $2 \cdot \#p^2 + 2 \cdot \#p + 2$ substituiert. Die Konstante c_2 wird gleich Eins gesetzt.

$$\begin{aligned} x &= 13 \cdot \#p^2 + 1 + 15 \cdot (2 \cdot \#p^2 + 2 \cdot \#p + 2) + \#p \\ &= 13 \cdot \#p^2 + 1 + (30 \cdot \#p^2 + 30 \cdot \#p + 30) + \#p \\ &= 43 \cdot \#p^2 + 31 \cdot \#p + 31 \end{aligned}$$

□

Verdopplung	Bitoperationen	Anzahl der Reduktionen
(29)	$2 \cdot \#p^2 + 1$	$2 \cdot A + B$
(30)	$4 \cdot \#p^2 + 2 \cdot \#p + 1$	$4 \cdot A + 3 \cdot B$
(31)	$\#p^2 + \#p + 1$	$A + 2 \cdot B$
(32)	–	–
(33)	$(\#p + \#p^2) + (\#p^2 + 1) + \#p$	$2 \cdot A + 3 \cdot B$
(34)	$\#p^2 + 1$	$A + B$
Summe	$10 \cdot \#p^2 + 5 \cdot \#p + 5$	$10 \cdot A + 10 \cdot B$

Die Verdopplung eines Punktes im Jacobi-Raum benötigt zur Berechnung der arithmetischen Operationen im worst case $10 \cdot \#p^2 + 5 \cdot \#p + 5$ Bitoperationen. Zusätzlich müssen 10 modulare Reduktionen mittels eines Reduktionsalgorithmus und 10 modulare Reduktionen mittels Subtraktion berechnet werden.

Satz 4.10 *Eine Punktverdopplung in der affinen Ebene ist günstiger als im Jacobi-Raum, wenn die Berechnung eines multiplikativen Inversen nicht teurer ist als* 13 *Multiplikationen ist.*

Beweis: Aus den Analysen der Punktverdopplung in der affinen Ebene und im Jacobi-Raum ergibt sich das Gleichungssystem:

$$x+3\cdot\#p^2+5\cdot\#p+3+4\cdot c_1\cdot\#p^2+7\cdot c_2\cdot\#p = 10\cdot\#p^2+5\cdot\#p+5+10\cdot c_1\cdot\#p^2+10\cdot c_2\cdot\#p$$

$$\begin{aligned}\Leftrightarrow x &= 7\cdot\#p^2+2+6\cdot c_1\cdot\#p^2+3\cdot c_2\cdot\#p\\ &\leq 13\cdot c_1\cdot\#p^2+3\cdot c_2\cdot\#p+2\end{aligned}$$

□

Für die Barrett-Reduktion und die Montgomery-Reduktion können die benötigten Bitoperationen detaillierter analysiert werden.

Satz 4.11 *Wird die Barrett-Reduktion verwendet, darf das Berechnen eines multiplikativen Inversen nicht mehr kosten als* 19 *Multiplikationen und* 33 *Additionen, damit die Punktaddition in der affinen Ebene günstiger ist als im Jacobi-Raum.*

Beweis: Die Reduktionen vom Typ A werden durch $2\cdot\#p^2+5\cdot\#p+8$ substituiert. Die Konstante c_2 wird gleich Eins gesetzt.

$$\begin{aligned}x &= 7\cdot\#p^2+2+6\cdot(2\cdot\#p^2+5\cdot\#p+8)+3\cdot\#p\\ &= 7\cdot\#p^2+2+(12\cdot\#p^2+30\cdot\#p+48)+3\cdot\#p\\ &= 19\cdot\#p^2+33\cdot\#p+50\end{aligned}$$

□

Satz 4.12 *Wird die Montgomery-Reduktion verwendet, darf das Berechnen eines multiplikativen Inversen nicht mehr kosten als* 19 *Multiplikationen und* 15 *Additionen, damit die Punktaddition in der affinen Ebene günstiger ist als im Jacobi-Raum.*

Beweis: Die Reduktionen vom Typ A werden durch $2\cdot\#p^2+2\cdot\#p+2$ substituiert. Die Konstante c_2 wird gleich Eins gesetzt.

$$\begin{aligned}x &= 7\cdot\#p^2+2+6\cdot(2\cdot\#p^2+2\cdot\#p+2)+3\cdot\#p\\ &= 7\cdot\#p^2+2+(12\cdot\#p^2+12\cdot\#p+12)+3\cdot\#p\\ &= 19\cdot\#p^2+15\cdot\#p+14\end{aligned}$$

□

Fazit: Die Analyse erweckt den Eindruck, dass bei Verwendung der Montgomery-Reduktion Punktoperationen auf elliptischen Kurven an schnellsten berechnet werden können. Bei der Analyse wurde jedoch nicht beachtet, dass nach einer Montgomery-Reduktion die Werte in Montgomeryform sind und wieder in Normalform zurückgerechnet werden müssen. Daraus folgt, dass die Barrett-Reduktion für Operationen auf elliptischen Kurven das geeignetere System ist.

4.3.4 Kurven in affiner Darstellung über $\mathbb{F}_{2^q}$

Für die Untersuchung der Punktverdopplung und der Punktaddition auf elliptischen Kurven über Körpern der Charakteristik 2 werden die gleichen Laufzeiten für die arithmetischen Operationen, die Bestimmung eines multiplikativen Inversen und die modulare Reduktion angenommen, wie bei elliptischen Kurven über Primkörpern.

Satz 4.13 *Zwei Polynome $a, b \in \mathbb{F}_{2^q}$ werden miteinander addiert, indem ihre Koeffizientenvektoren komponentenweise Modulo 2 addiert werden.*

Korollar 4.1 *Die Summe s zweier Polynome $s = a+b$; $a, b \in \mathbb{F}_{2^q}$ muss niemals modular reduziert werden.*

Für die Punktaddition und die Punktverdopplung auf elliptischen Kurven über Körpern der Charakteristik 2 muss, wie bei Primkörpern, ein multiplikatives Inverses bestimmt werden. Jedoch müssen weniger arithmetische Operationen durchgeführt werden als im projektiven Raum.

Addition	Bitoperationen	Anzahl der Reduktionen
(35)	$\#p^3 + \#p^2 + 2 \cdot \#p$	A
(36)	$\#p^2 + 4 \cdot \#p$	A
(37)	$\#p^2 + 3 \cdot \#p$	A
Summe:	$\#p^3 + 3 \cdot \#p^2 + 9 \cdot \#p$	$3 \cdot A$

Die Punktaddition von zwei Punkten $P, Q \neq \mathcal{O}$ auf einer elliptischen Kurve über einem Körper der Charakteristik 2 benötigt für die arithmetischen Operationen $\#p^3 + 3 \cdot \#p^2 + 9 \cdot \#p$ Bitoperationen und 3 modulare Reduktionen mittels eines Reduktionsalgorithmus.

Verdopplung	Bitoperationen	Anzahl der Reduktionen
(38)	$\#p^3 + \#p^2 + \#p$	A
(39)	$\#p^2 + 2 \cdot \#p$	A
(40)	$\#p^2 + 3 \cdot \#p$	A
Summe:	$\#p^3 + 3 \cdot \#p^2 + 6 \cdot \#p$	$3 \cdot A$

Für die Punktverdopplung eines Punktes $P \neq \mathcal{O}$ auf einer elliptischen Kurve über einen Körper der Charakteristik 2 werden für die arithmetischen Operationen $\#p^3 + 3 \cdot \#p^2 + 6 \cdot \#p$ Bitoperationen benötigt. Zusätzlich müssen 3 modulare Reduktionen mittels eines Reduktionsalgorithmus berechnet werden.

4.3.5 Kurven in projektiver Darstellung über $\mathbb{F}_{2^q}$

Die Punktadditionen und die Punktverdopplungen auf elliptischen Kurven über Körpern der Charakteristik 2 im projektiven Raum benötigen keine Berechnung eines multiplikativen Inversen, jedoch müssen mehr arithmetische Operationen durchgeführt werden.

Addition	Bitoperationen	Anzahl der Reduktionen
(41)	$2 \cdot \#p^2$	$2 \cdot A$
(42)	$2 \cdot \#p^2$	$2 \cdot A$
(43)	$\#p$	–
(44)	$2 \cdot \#p^2$	$2 \cdot A$
(45)	$2 \cdot \#p^2$	$2 \cdot A$
(46)	$\#p$	–
(47)	$\#p^2$	A
(48)	$2 \cdot \#p^2 + \#p$	$2 \cdot A$
(49)	$\#p^2$	A
(50)	$\#p$	–
(51)	$5 \cdot \#p^2 + 2 \cdot \#p$	$5 \cdot A$
(52)	$3 \cdot \#p^2 + \#p$	$3 \cdot A$
Summe:	$20 \cdot \#p^2 + 7 \cdot \#p$	$20 \cdot A$

Für die Punktaddition zweier Punkte $P, Q \neq \mathcal{O}$ auf einer elliptischen Kurve über einem Körper der Charakteristik 2 werden für die arithmetischen Berechnungen $20 \cdot \#p^2 + 7 \cdot \#p$ Bitoperationen benötigt. Weitere 20 modulare Reduktionen müssen mittels eines Reduktionsalgorithmus bestimmt werden.

Satz 4.14 *Damit eine Punktaddition zweier Punkte auf einer elliptischen Kurve über einem Körper der Charakteristik 2 in der affinen Ebene im worst case die gleichen Kosten wie im projektiven Raum verursacht, darf die Bestimmung eines multiplikativen Inversen nicht mehr kosten als* 34 *Multiplikationen.*

Beweis: Aus der Analyse der Punktaddition in der affinen Ebene und im projektiven Raum auf elliptischen Kurven über Körpern der Charakteristik 2 ergibt das folgenden Gleichungssystem:

$$\begin{aligned} x + 3 \cdot \#p^2 + 9 \cdot \#p + 3 \cdot c_1 \#p^2 &= 20 \cdot \#p^2 + 7 \cdot \#p + 20 \cdot c_1 \#p^2 \\ \Leftrightarrow x &= 17 \cdot \#p^2 - 2 \cdot \#p + 17 \cdot c_1 \cdot \#p^2 \\ &\leq 34 \cdot c_1 \#p^2 - 2 \cdot \#p \end{aligned}$$

Wenn das Bestimmen eines multiplikativen Inversen in einem endlichen Körper der Charakteristik 2 keine höheren Kosten verursacht als 34 Multiplikationen, ist eine Punktaddition auf einer elliptischen Kurve über einem Körper der Charakteristik 2 in der affinen Ebene günstiger als im projektiven Raum.

□

4.4 Laufzeitunterschiede zwischen den verschiedenen skalaren Multiplikationsalgorithmen

Um die Algorithmen zur skalaren Multiplikation miteinander vergleichen zu können, werden die worst cases und die average cases als Eingabe für die Algorithmen betrachtet. Für die Analyse der worst cases werden Skalare der Bitlänge n so angegeben, dass die benötigten Operationen auf der Kurve maximal sind. Für die Analyse der average cases werden Skalare in Binärdarstellung mit Hamming-Gewicht $\frac{1}{2}$ betrachtet, da diese Faktoren statistisch am häufigsten auftreten.
Für den Vergleich wird die skalare Multiplikation $[k] \cdot P$ betrachtet. Die Länge der Binärdarstellung von k sei als n definiert.

4.4.1 Analyse der worst cases

Der worst case für die Multiplizierer durch wiederholtes Verdoppeln ist, wenn die Binärdarstellung des Skalars nur aus Einsen besteht; d.h. $k = 2^n - 1,\ n \in \mathbb{N}$. Der Algorithmus 5 zur skalaren Multiplikation von rechts nach links benötigt keine Umrechnung des Faktors in eine NAF oder Decomposition. In jedem der n Schleifendurchläufe wird eine Verdopplung des Punktes P berechnet. Wenn in der Binärdarstellung des Skalars eine 1 auftritt, wird eine weitere Punktaddition berechnet. Da im worst case das Skalar nur aus Einsen besteht, müssen n Punktadditionen berechnet werden. Die erste Addition kann vernachlässigt werden, da das Vielfache von P mit dem neutralen Element addiert wird. Daraus ergeben sich:

$$\begin{array}{cc} \text{Verdopplungen} & n \\ \text{Additionen} & n-1 \end{array}$$

Algorithmus 6, der durch wiederholtes Verdoppeln von links nach rechts das Produkt $[k] \cdot P$ berechnet, benötigt keine Vorberechnung. In jedem Schleifendurchlauf muss im worst case eine Punktverdopplung und eine weitere Punktaddition berechnet werden, da der Skalar nur aus Einsen besteht. Die erste Verdopplung und die erste Punktaddition kann vernachlässigt werden, da das neutrale Element der Addition verdoppelt und dann mit P addiert wird; $2 \cdot \mathcal{O} = \mathcal{O}$ und $P + \mathcal{O} = P$.
Daraus ergeben sich:

$$\begin{array}{cc} \text{Verdopplungen} & n-1 \\ \text{Additionen} & n-1 \end{array}$$

Die NAF eines Skalars k ist maximal ein Bit länger als die Binärdarstellung von k. Für jede NAF $(s_j s_{j-1} \cdots s_1 s_0),\ s_i \in \{(-1), 0, 1\}$ gilt $s_i \cdot s_{i-1} = 0,\ s_i < 0$. Damit ist das Hamming-Gewicht eines Skalars nach oben beschränkt durch $\frac{n}{2}+1$. Die NAF kann in $O(n)$ berechnet werden. Damit müssen im worst case

$$\begin{array}{cc} \text{Verdopplungen} & n+1 \\ \text{Additionen} & \frac{n}{2}+1 \end{array}$$

berechnet werden, wenn ein Rechts-Links-Multiplizierer durch wiederholtes Verdoppeln verwendet wird. Wird ein Links-Rechts-Multiplizierer durch wiederholtes Verdoppeln verwendet, so müssen im worst case

$$\begin{array}{cc} \text{Verdopplungen} & n \\ \text{Additionen} & \frac{n}{2}+1 \end{array}$$

berechnet werden.
Die Effizienz des m-ären Sliding-Window-Multiplizierers ist abhängig von der gewählten Fenstergröße r. Der worst case für diese Eingabe ist, wenn der Skalar in $\lceil \frac{n}{r} \rceil = d$ Fenster eingeteilt wird; d.h. das zwischen den einzelnen Fenstern keine Blöcke aus Nullen sind. Die Berechnung der Decomposition benötigt $O(n)$ Bitoperationen. Die Decomposition wird dem Algorithmus 10 übergeben. Bei der Vorberechnung werden immer eine Punktverdopplung und $2^{r-2} - 1$ Punktadditionen berechnet. In der Schleife des Hauptprogrammes werden $n - (n \mod r)$ Punktverdopplungen und weitere d Punktadditionen berechnet.
Daraus ergeben sich:

$$\begin{array}{cc} \text{Verdopplungen} & 1 + (n - (n \mod r)) \\ \text{Additionen} & (2^{r-2} - 1) + d \end{array}$$

4.4.2 Analyse der average cases

Im average case haben Skalare das Hamming-Gewicht $\frac{1}{2}$. Die Anzahl der benötigten Punktadditionen bei skalaren Multiplikationen mittels wiederholter Verdopplung ist abhängig von dem Hamming-Gewicht. Die Anzahl der Punktverdopplungen ist bei dem Rechts-Links-Multiplizierer immer n und beim Links-Rechts-Multiplizierer immer $n-1$. Daher sind beim Hamming-Gewicht $\frac{1}{2}$ die Anzahl der Operationen beim Rechts-Links-Multiplizierer

$$\begin{array}{cc} \text{Verdopplungen} & n \\ \text{Additionen} & \frac{n}{2}-1 \end{array}$$

und beim Links-Rechts-Multiplizierer

$$\begin{array}{cc} \text{Verdopplungen} & n-1 \\ \text{Additionen} & \frac{n}{2}-1 \end{array}$$

Für den NAF-Multiplizierer ist die Anzahl von Punktverdopplungen und Punktadditionen abhängig von der Positionierung der Einsen und Nullen der Binärdarstellung des Skalars k. Gilt für die Binärdarstellung des Skalars: $k_i \neq k_{i-1},\ i \in \mathbb{N},\ k_j \in \{0,1\}$, so ist das Skalar eine NAF. Existieren in der Binärdarstellung Blöcke aus Einsen und Nullen, dann ist das Hamming-Gewicht der NAF kleiner oder gleich dem Hamming-Gewicht der Binärdarstellung. Für eine NAF ist das Hamming-Gewicht im average case $\frac{1}{3}$ [5]. Die NAF des Skalars k ist maximal ein Bit länger als die Binärdarstellung von k. Somit müssen im average case

$$\begin{array}{cc} \text{Verdopplungen} & n+1 \\ \text{Additionen} & \frac{n}{3} \end{array}$$

mit einem Rechts-Links-Multiplizierer berechnet werden. Wird ein Links-Rechts-Multiplizierer verwendet müssen

$$\begin{array}{cc} \text{Verdopplungen} & n \\ \text{Additionen} & \frac{n}{3} \end{array}$$

berechnet werden.
Im average case hat die Decomposition eines Skalars $(r-1)n/(r+1)$ Werte ungleich Null [5]. Während der Berechnung der Decomposition kann sich die Länge des Skalars um ein Bit erhöhen. Nach der Arbeit von M. Otto müssen im average case im Hauptprogramm $\frac{1}{3} \cdot \lceil \frac{n}{d} \rceil + (\frac{2}{3} - \frac{4}{3 \cdot 2^d}) \cdot \lceil \frac{n+1}{d} \rceil - \frac{2^d - \frac{5}{3}}{2^d}$ Punktadditionen berechnet werden [18]. Für den m-ären Sliding-Window-Multiplizierer ergeben sich im average case daraus

$$\begin{array}{cc} \text{Verdopplungen} & 1 + (n - (n \mod r)) \\ \text{Additionen} & (2^{r-2} - 1) + \frac{1}{3} \cdot \lceil \frac{n}{d} \rceil + (\frac{2}{3} - \frac{4}{3 \cdot 2^d}) \cdot \lceil \frac{n+1}{d} \rceil - \frac{2^d - \frac{5}{3}}{2^d} \end{array}$$

für die skalare Multiplikation des Punktes P mit einem Skalar k der binären Länge n.
Fazit: Nur bei den Multiplizierern, die direkt auf der Binärdarstellung des Skalars rechnen, sind die benötigten Operationen auf einer elliptischen Kurve abhängig vom Hamming-Gewicht des Skalars. Die Effizienz des NAF-Multiplizierers und des m-ären Sliding-Window-Multiplizierers ist abhängig von der Positionierung der Einsen und Nullen in der Binärdarstellung des Skalars. In der Praxis und in Zufallsexperimenten wird erwartet, dass das Hamming-Gewicht

eines Faktors bei $\frac{1}{2}$ liegt. Die Statistiken der Zufallsexperimente sind im Anhang abgebildet. Hier werden pseudozufällig 50 elliptische Kurven, Punkte auf den Kurven und Skalare generiert und deren Multiplikationen berechnet. Die Körper der elliptischen Kurven haben binäre Längen zwischen 32 und 239 Bit. Über die Anzahl der Operationen wird der Durchschnitt gebildet und visualisiert.

5 Überblick

Koordinatensysteme über $\mathbb{F}_p$ und $\mathbb{F}_{2^q}$:
Betrachtet werden Körper der binären Länge $\#p$. A bezeichnet modulare Reduktionen mittels Reduktionsalgorithmus und B bezeichnet modulare Reduktionen mittels Subtraktion.

	Affin	Projektiv	Jacobi
Addition $\mathbb{F}_p$	$\#p^3 + 3 \cdot \#p^2 + 6 \cdot \#p + 3 \cdot A + 6 \cdot B$	$17 \cdot \#p^2 + 6 \cdot \#p + 1 + 17 \cdot A + 7 \cdot B$	$16 \cdot \#p^2 + 6 \cdot \#p + 1 + 18 \cdot A + 7 \cdot B$
Verdopplung $\mathbb{F}_p$	$\#p^3 + 4 \cdot \#p^2 + 5 \cdot \#p + 3 + 4 \cdot A + 7 \cdot B$	$13 \cdot \#p^2 + 5 \cdot \#p + 6 + 13 \cdot A + 10 \cdot B$	$10 \cdot \#p^2 + 5 \cdot \#p + 5 + 10 \cdot A + 10 \cdot B$
Addition $\mathbb{F}_{2^q}$	$\#p^3 + 3 \cdot \#p^2 + 9 \cdot \#p + 3 \cdot A$	$20 \cdot \#p^2 + 7 \cdot \#p + 20 \cdot A$	–
Verdopplung $\mathbb{F}_{2^q}$	$\#p^3 + 3 \cdot \#p^2 + 6 \cdot \#p + 3 \cdot A$	$10 \cdot \#p^2 + 4 \cdot \#p + 11 \cdot A$	–

Reduktionen vom Typ A:

	Standardreduktion	Barrett-Reduktion	Montgomery-Reduktion
Bitoperationen	$2 \cdot (\#x \cdot \#p - \#p^2) + \#x + \#p + 1$	$(\#x - \#p + 4) \cdot \#p + (\#x - \#p) \cdot \#\mu + 8$	$2 \cdot \#p^2 + 2 \cdot \#p + 2$

Skalare Multiplikationsalgorithmen:

		Rechts-Links-Multiplizierer	Links-Rechts-Multiplizierer	NAF-Multiplizierer (RL, LR)	m-ärer Sliding-Window-Multiplizierer
worst case	Additionen	$\#p - 1$	$\#p - 1$	$(\#p/2) + 1$, $(\#p/2) + 1$	$(2^{r-2} - 1) + d$
	Verdopplungen	$\#p$	$\#p - 1$	$\#p + 1$, $\#p$	$1 + (\#p - (\#p \mod r))$
average case	Additionen	$(\#p/2) - 1$	$(\#p/2) - 1$	$\#p/3$, $\#p/3$	$(2^{r-2} - 1) + \frac{1}{3} \cdot \lceil \frac{\#p}{d} \rceil + (\frac{2}{3} - \frac{4}{3 \cdot 2^d}) \cdot \lceil \frac{\#p+1}{d} \rceil - \frac{2^d - \frac{5}{3}}{2^d}$
	Verdopplungen	$\#p$	$\#p - 1$	$\#p + 1$, $\#p$	$1 + (\#p - (\#p \mod r))$

6 Implementierung

Im folgenden Abschnitt werden Besonderheiten beschrieben, die bei der Implementierung beachtet werden müssen. Zudem wird das *Graphical User Interface*, kurz GUI, vorgestellt und die Funktionen beschrieben.

6.1 Elliptische Kurven über Primkörpern

6.1.1 Die Implementierung

Bei der Implementierung der Algorithmen zur Punktaddition und Punktverdopplung müssen einige Besonderheiten beachtet werden, die nicht in der allgemeinen Literatur erwähnt werden. In der schwer zugänglichen *IEEE 1363*, einer internationalen Standardisierung zur ECC, sind diese Besonderheiten beschrieben [14]. Soll ein Punkt auf einer elliptischen Kurve verdoppelt werden, dessen y-Koordinate gleich Null ist, ist die Verdopplung das neutrale Element der Addition.
Für den NAF-Multiplizierer und für den m-ären Sliding-Window-Multiplizierer muss das additive Inverse von Punkten auf einer elliptischen Kurve bestimmt werden. Bei elliptischen Kurven über Primkörpern ist die Bestimmung des Inversen eines Punktes kostengünstig. Im projektiven Raum und im Jacobi-Raum hat jeder Punkt auf einer elliptischen Kurve $p-1$ Repräsentationen. Im projektiven Raum sind zwei Punkte $P, P' \in E_{\mathbb{F}_p}$ gleich, wenn ein $t \in \mathbb{F}_p$ existiert, sodass $P = (x, y, z) = (tx, ty, tz) = P'$. Im Jacobi-Raum sind zwei Punkte $P, P' \in E_{\mathbb{F}_p}$ gleich, wenn ein $t \in \mathbb{F}_p$ existiert, sodass $P = (x, y, z) = (t^2x, t^3y, tz) = P'$. Um festzustellen, ob zwei Punkte gleich sind, kann ein multiplikatives Inverses bestimmt werden. Das Berechnen eines multiplikativen Inversen würde die Vorteile von elliptischen Kurven im projektiven Raum und im Jacobi-Raum aufheben. In der Praxis werden zwei Punkte P und Q mit unterschiedlichen Koordinaten immer zuerst miteinander addiert. Ist die Summe ein Nullvektor, wird einer der Punkte verdoppelt.

Satz 6.1 *Seien P und Q zwei Punkte auf einer elliptischen Kurve in projektiver Darstellung. Gilt $P = (x, y, z)$ und $Q = (tx, ty, tz)$, $t \in \mathbb{F}_p^*$, $x, y, z \in \mathbb{F}_p$, so ist das Ergebnis des Algorithmus zur Addition* $(0, 0, 0)$.

Beweis: Gegeben seien zwei Punkte P und Q auf einer elliptischen Kurve im projektiven Raum. P sei definiert als $P = (x, y, z) = (X_1, Y_1, Z_1)$ und Q sei definiert als $Q = (tx, ty, tz) = (X_2, Y_2, Z_2)$

$$\begin{aligned}
u &:= Y_2Z_1 - Y_1Z_2 = (ty)z - y(tz) = 0 \\
v &:= X_2Z_1 - X_1Z_2 = (tx)z - x(tz) = 0 \\
A &:= (u^2Z_1Z_2 - v^3) - (2v^2X_1Z_2) = (0 \cdot z(tz) - 0^3) - (2 \cdot 0^2x(tz)) = 0 \\
X_3 &:= vA = 0 \cdot 0 = 0 \\
Y_3 &:= (v^2X_1Z_2 - A)u - (v^3Y_1Z_2) = (0^2 \cdot x(tz) - 0) \cdot 0 - (0^3 \cdot y(tz)) = 0 \\
Z_3 &:= v^3Z_1Z_2 = 0^3z(tz) = 0
\end{aligned}$$

$(X_3, Y_3, Z_3) := (0, 0, 0)$ □

Satz 6.2 *Seien P und Q zwei Punkte auf einer elliptischen Kurve in Jacobidarstellung. Gilt $P = (x, y, z)$ und $Q = (t^2x, t^3y, tz)$, $t \in \mathbb{F}_p^*$, $x, y, z \in \mathbb{F}_p$, so ist das Ergebnis des Algorithmus zur Addition* $(0, 0, 0)$.

Beweis: Gegeben seien zwei Punkte P und Q auf einer elliptischen Kurve im Jacobi-Raum. P sei definiert als $P = (x, y, z)$ und Q sei definiert als $Q = (t^2x, t^3y, tz)$

$$\begin{aligned}
u_1 &:= X_1Z_2^2 = x(t^2z^2)\\
u_2 &:= X_2Z_1^2 = (t^2x)z^2\\
s_1 &:= Y_1Z_2^3 = y(t^3z^3)\\
s_2 &:= Y_2Z_1^3 = (t^3y)z^3\\
H &:= u_2 - u_1 = x(t^2z^2) - (t^2x)z^2 = 0\\
r &:= s_2 - s_1 = y(t^3z^3) - (t^3y)z^3 = 0\\
X_3 &:= (r^2 - H^3) - 2u_1H^2 = (0^2 - 0^3) - 2u_10^2 = 0\\
Y_3 &:= r(u_1H^2 - X_3) - s_1H^3 = 0(u_10^2 - 0) - s_1 \cdot 0^3 = 0\\
Z_3 &:= Z_1Z_2H = z \cdot tz \cdot 0 = 0\\
(X_3, Y_3, Z_3) &:= (0, 0, 0)
\end{aligned}$$

□

Der Punkt $(0, 0, 0)$ ist im projektiven und im Jacobi-Raum nicht definiert. Diese Koordinate entspricht dem *Standpunkt des Betrachters*, wo kein Punkt auf der elliptischen Kurve liegen kann.

6.1.2 Die GUI

Die GUI ist in zwei Karteikarten oder *Tabs* unterteilt. Das initiale Tab ist für die Analyse von elliptischen Kurven über Primkörpern aufgebaut (Abbildung 10). Das Tab unterteilt sich in zwei Bereiche. Im oberen Teil kann das Koordinatensystem und die elliptische Kurve festgelegt werden. Mit der *Spinbox* kann die binäre Länge des Primkörpers festgelegt werden. Der Button *Generate* generiert mit der NTL-Funktion `GenPrime_ZZ(`*long*`)` eine pseudozufällige Primzahl `_prime` der in der Spinbox angegebenen binären Länge; die Koeffizienten a und b werden nicht pseudozufällig erzeugt. Pseudozufällige Zahlen werden in dieser Arbeit immer mit den Funktionen der NTL erzeugt. Mit dem Button *Random* wird eine pseudozufällige Primzahl der in der Spinbox angegebenen binären Länge und die Koeffizienten a und b pseudozufällig erzeugt. Die Koeffizienten werden mit der NTL-Funktion `RandomBnd(_prime)` generiert. Mit dem *OK*-Button wird die angegebene elliptische Kurve bestätigt und das untere Feld freigegeben.

In dem *Addition*-Tab können zwei Punkte P und Q auf der angegebenen elliptischen Kurve miteinander addiert werden (Abbildung 11). Mit den *Random*-Buttons können pseudozufällige Punkte auf der elliptischen Kurve festgelegt werden. Dazu wird eine pseudozufällige x-Koordinate X aus dem Körper $\mathbb{F}_{\texttt{\_prime}}$, mit den Funktionen der NTL generiert und dazu die y-Koordinate berechnet;

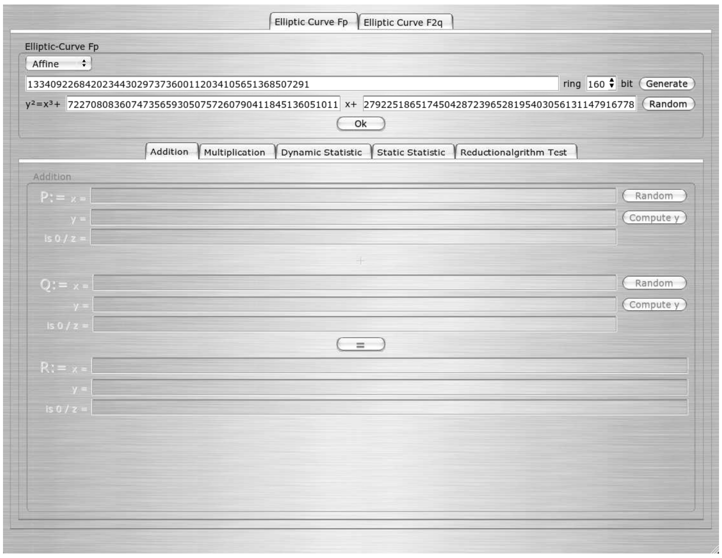

Abbildung 10: Körper und Koordinatensystemwahl

sollte es keinen Punkt auf der elliptischen Kurve mit der x-Koordinate X geben, so wird ein neues X erzeugt. Dieser Vorgang wird solange wiederholt, bis ein Punkt auf der elliptischen Kurve gefunden ist; der Punkt $\mathcal{O}$ wird nie erzeugt. Wird die x-Koordinate eines Punktes per Hand angegeben, so kann die y-Koordinate mit dem *"Compute y"*-Button berechnet werden. Wenn es keinen Punkt auf der elliptischen Kurve mit der gegebenen x-Koordinate gibt, wird der Punkt $\mathcal{O}$ zurückgegeben. Bei der Rückgabe eines Punktes wird jede der beiden möglichen y-Koordinaten mit Wahrscheinlichkeit $\frac{1}{2}$ ausgewählt; ob $y = 0$ gilt wird dabei nicht beachtet. Mit dem "="-Button wird die Summe von P und Q berechnet.

In dem *Multiplication*-Tab kann das skalare Produkt eines Punktes P auf der festgelegten elliptischen Kurve berechnet werden (Abbildung 12). Dazu kann mit dem oberen *Random*-Button ein pseudozufälliges Skalar generiert werden; ein Skalar kann aber auch per Hand eingegeben werden. In dem *Dropdown*-Menü kann der zu verwendende Algorithmus zur skalaren Multiplikation ausgewählt werden. Wird ein Algorithmus gewählt, der das Skalar in Fenster einteilt, kann die Fenstergröße in einer Spinbox eingestellt werden; auf die anderen Algorithmen hat die Spinbox keine Auswirkungen. Wie im Additions-Tab kann ein Punkt pseudozufällig generiert werden. Per Hand kann die x-Koordinate eines Punktes angegeben werden und mit dem "Compute y"-Button die dazugehörige y-Koordinate berechnet werden. Mit dem "="-Button wird das skalare Produkt mittels des gewählten Algorithmus berechnet. Soll ein Punkt auf einer elliptischen Kurve im projektiven oder im Jacobi-Raum vervielfältigt werden, so wird das Produkt durch das Drücken auf den *"Affine Coord."*-Button in die affine Darstellung umgerechnet.

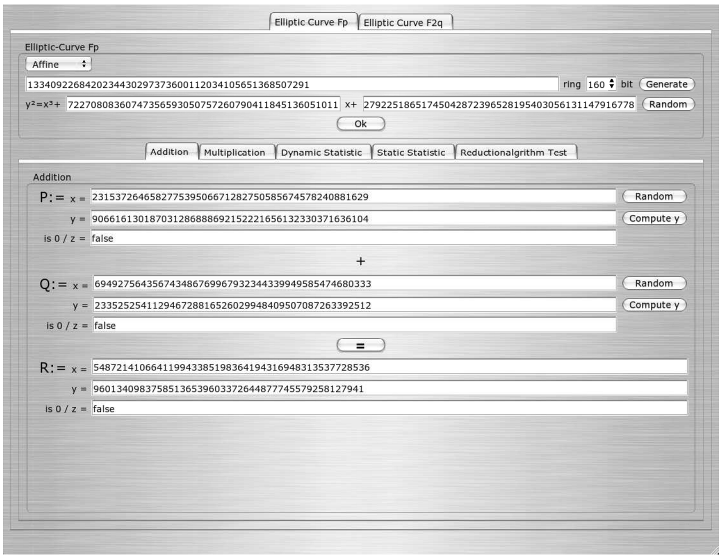

Abbildung 11: Punktaddition

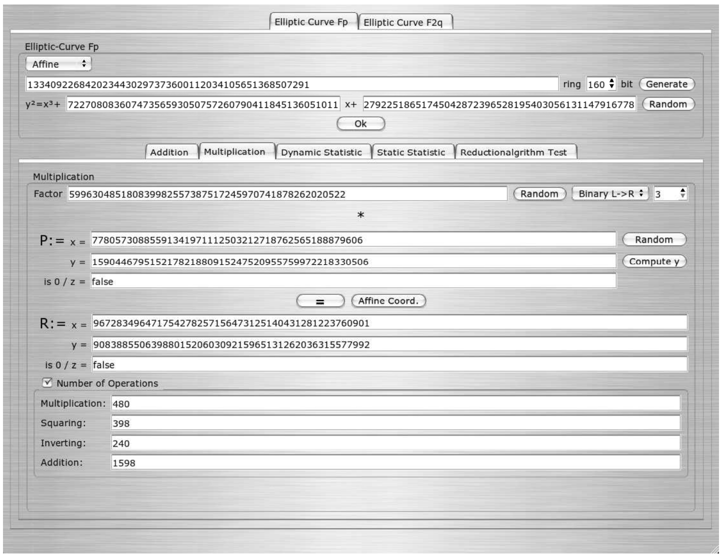

Abbildung 12: skalare Multiplikation

Das *Number of Operations*-Feld kann durch Anhaken aktiviert werden. Hier werden die arithmetischen Operationen mitgezählt, die bei der Berechnung des skalaren Produktes benötigt werden. Die Operationen werden nur dann mitgezählt, wenn das Feld aktiviert ist. Dies wird durch eine Unterscheidung in der Implementierung erzielt.
In dem *"Dynamic Statistic"*-Tab werden pseudozufällig elliptische Kurven in den verfügbaren Koordinatensystemen erzeugt und auf diesen Kurven skalare Multiplikationen berechnet (Abbildung 13). Die im oberen Bereich angegebene elliptische Kurve wird nicht beachtet. Die Anzahl der skalaren Multiplikationen auf einer elliptischen Kurve kann in einer Spinbox angegeben werden. Die binäre Größe der Primkörper wird in dem Bereich von 32-Bit bis 240-Bit angegeben. In dem *Dropdown*-Menü kann der zu verwendende Algorithmus zur skalaren Multiplikation ausgewählt werden. Mit dem *"Start Test"*-Button wird der Testlauf gestartet. Über die Anzahl der arithmetischen Operationen und der Kurvenoperationen wird der Durchschnitt berechnet und in grafischer Form dargestellt.

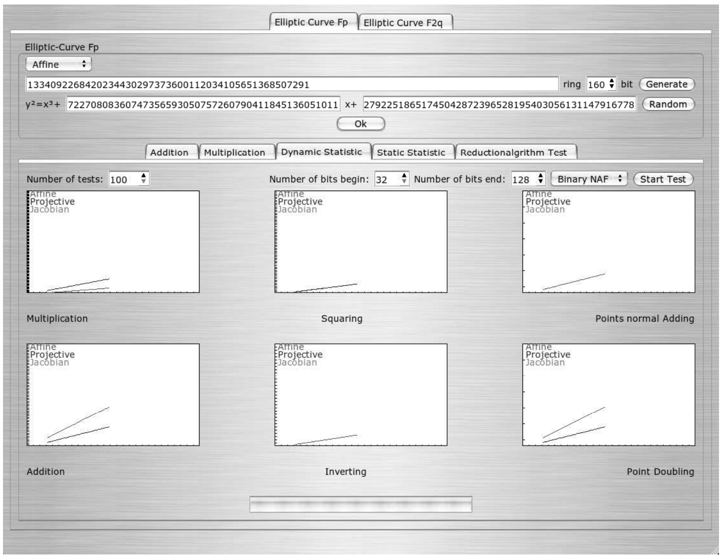

Abbildung 13: Dynamischer Test

In dem *"Static Statistic"*-Tab können zu einer vorgegebenen elliptischen Kurve in einem festen Koordinatensystem die verschiedenen Algorithmen zur skalaren Multiplikation getestet werden (Abbildung 14). Für die Tests werden pseudozufällige Punkte auf der elliptischen Kurve und Faktoren generiert. Die Anzahl der Punktgenerationen und Skalargenerationen kann in einer Spinbox festgelegt werden. Neben der Spinbox können die Algorithmen zur skalaren Multiplikation ausgewählt werden. Mit dem *"Start Test"*-Button wird der Testlauf gestartet. Die Anzahl der arithmetischen Operationen und der Kurvenoperationen werden in den Graphen einzeln dargestellt; dadurch ergeben sich Punktwolken die die Streuung der Operationen beschreiben (Abbildung 15).

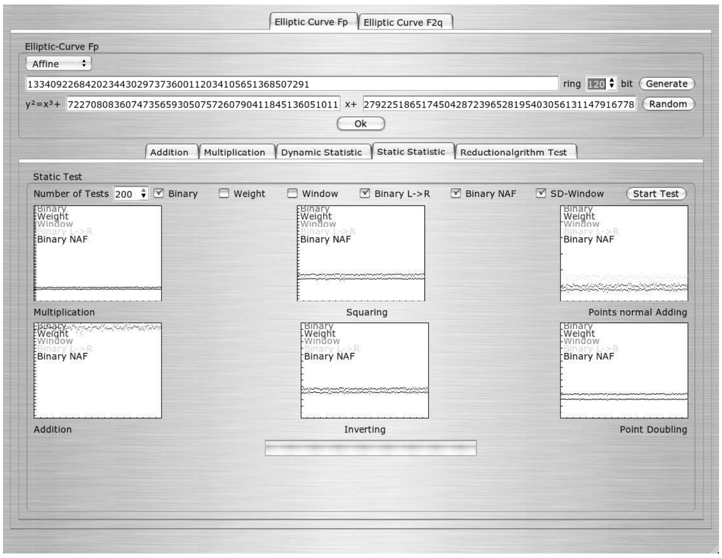

Abbildung 14: Statischer Test

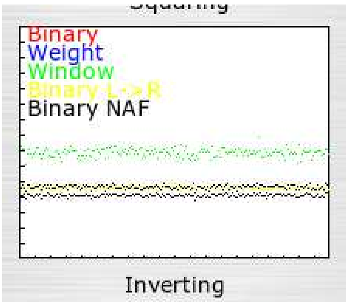

Abbildung 15: Punktwolken der Inversionen bei skalaren Multiplikationen

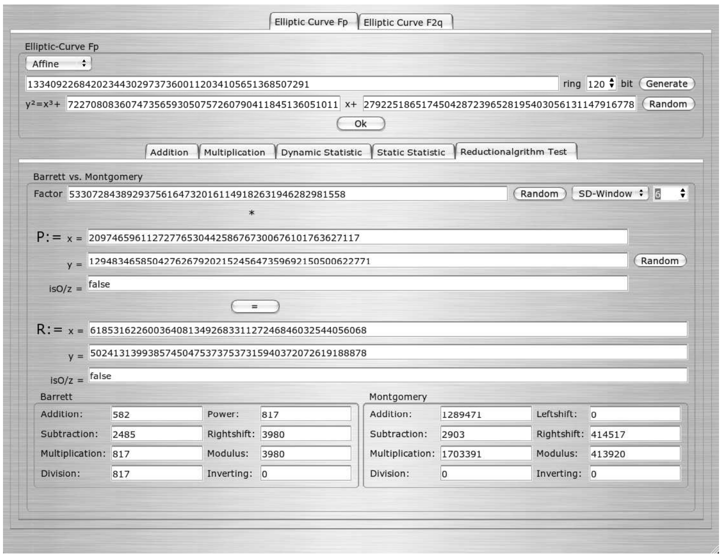

Abbildung 16: Reduktionsalgorithmenvergleich

Ein Teil der Studienarbeit ist es die Einsetzbarkeit der verschiedenen Reduktionsalgorithmen anhand der benötigten Operationen zu vergleichen. Dafür ist das *"Reduktionsalgorithm Test"*-Tab eingefügt worden (Abbildung 16). In diesem Tab kann eine skalare Multiplikation berechnet werden, wobei die arithmetischen Operationen und die Bitoperationen bei den modularen Reduktionen mitgezählt werden. Dieser Tab ist genauso aufgebaut wie das *Multiplikations*-Tab, jedoch werden im unteren Bereich die Operationen, die zur modularen Reduktion mittels Barrett- und Montgomery-Reduktion benötigt werden, angezeigt.

Der Test wird nach Absprache nur für das affine Koordinatensystem implementiert, da die Montgomery-Reduktion und die Montgomery-Multiplikation bereits in diesem Koordinatensystem ein Vielfaches der Operationen benötigt, die die Barrett-Reduktion braucht. Dies liegt daran, dass bei den Algorithmen von Montgomery die Zwischenergebnisse in Montgomeryform gehalten werden müssen. Bei den Punktadditionen und den Punktverdopplungen müssen viele arithmetische Operationen durchgeführt und modular reduziert werden. Viele Zwischenergebnisse sind in Normalform und viele Endergebnisse sind in Montgomeryform. Diese müssen jeweils umgerechnet werden. Dieser häufige Wechsel zwischen den verschiedenen Formen erhöht die Anzahl der verschiedenen arithmetischen Operationen stark.

6.2 Elliptische Kurven über Körpern der Charakteristik 2

6.2.1 Die Implementierung

Wie bei elliptischen Kurven über Primkörpern gibt es bei elliptischen Kurven über Körpern der Charakteristik 2 Besonderheiten, die bei der Implementierung beachtet werden müssen. Soll ein Punkt verdoppelt werden, dessen x-Koordinate gleich Null ist, ist die Verdopplung des Punktes das neutrale Element der Addition. Zwei Punkte P, Q auf einer elliptischen Kurve über einem Körper der Charakteristik 2 sind im projektiven Raum gleich, wenn ein $t \in \mathbb{F}_{2^q}$ existiert, sodass $P = (x, y, z) = (t^2x, t^3y, tz) = Q$. Um herauszufinden, ob zwei Punkte gleich sind, kann das multiplikative Inverse von z berechnet werden und damit t bestimmt werden. In der Praxis werden jedoch immer zuerst zwei Punkte mit unterschiedlichen Koordinaten miteinander addiert. Wenn die Summe zweier Punkte mit unterschiedlichen Koordinaten im projektiven Raum als Lösung einen Nullvektor ergibt, so müssen die Punkte gleich sein. Dann muss einer der Punkte verdoppelt werden.

Satz 6.3 *Seien P und Q zwei Punkte auf einer elliptischen Kurve über einem Körper der Charakteristik 2 in projektiver Darstellung. Gilt $P = (x, y, z)$ und $Q = (t^2x, t^3y, tz)$, $t, x, y, z \in \mathbb{F}_{2^q}$, so ist das Ergebnis des Algorithmus zur Addition* $(0, 0, 0)$.

Beweis: Gegeben seien zwei Punkte $P = (x, y, z) = (X_1, Y_1, Z_1)$ und $P = (t^3x, t^2y, tz) = (X_2, Y_2, Z_2)$ einer elliptischen Kurve in projektiver Darstellung, $t, x, y, z \in \mathbb{F}_{2^q}, t \neq 0$.

$$
\begin{aligned}
\lambda_1 &:= X_1Z_2^2 = x(t^2z^2) \\
\lambda_2 &:= X_2Z_1^2 = (t^2x)z^2 \\
\lambda_3 &:= \lambda_1 + \lambda_2 = x(t^2z^2) + (t^2x)z^2 = 0 \\
\lambda_4 &:= Y_1Z_2^3 = x(t^3z^3) \\
\lambda_5 &:= Y_2Z_1^3 = (t^3x)z^3 \\
\lambda_6 &:= \lambda_4 + \lambda_5 = x(t^3z^3) + (t^3x)z^3 = 0 \\
\lambda_7 &:= Z_1\lambda_3 = z \cdot 0 = 0 \\
\lambda_8 &:= \lambda_6X_2 + \lambda_7Y_2 = 0 \cdot t^2x + 0 \cdot t^3y = 0 \\
Z_3 &:= \lambda_7Z_2 = 0 \cdot tz = 0 \\
\lambda_9 &:= \lambda_6 + Z_3 = 0 + 0 = 0 \\
X_3 &:= aZ_3^2 + \lambda_6\lambda_9 + \lambda_3^3 = a \cdot 0^2 + 0 \cdot 0 + 0^3 = 0 \\
Y_3 &:= \lambda_9X_3 + \lambda_8\lambda_7^2 = 0 \cdot 0 + 0 \cdot 0^2 = 0
\end{aligned}
$$

$(X_3, Y_3, Z_3) := (0, 0, 0)$ $\square$

Um das additive Inverse eines Punktes auf einer elliptischen Kurve über einem Primkörper zu invertieren, muss die y-Koordinate des Punktes negiert werden. Für einen Punkt $P = (x, y)$ in der affinen Ebene gilt: $-P = (x, -y)$. Die Punktinversion im projektiven Raum und im Jacobi-Raum ist analog. Das additive Inverse eines Punktes auf einer elliptischen Kurve über einem Körper der Charakteristik 2 kann nicht auf diese Weise bestimmt werden. Das additive Inverse eines Punktes $P = (x, y)$ in affiner Darstellung auf einer elliptischen Kurve über einem Körper der Charakteristik 2 ist $-P = (x, x + y)$. Das additive Inverse eines Punktes $P = (x, y, z)$ in projektiver Darstellung auf einer elliptischen Kurve über einem Körper der Charakteristik 2 ist $-P = (x, xz + y, z)$.

6.2.2 Die GUI

Mit dem zweiten *Haupt-Tab* (Abbildung 17) können Berechnungen auf elliptischen Kurven über Körpern der Charakteristik 2 durchgeführt werden. In dem ersten Menü kann das Koordinatensystem ausgewählt werden, auf dem gerechnet werden soll. Soll ein irreduzibles Polynom pseudozufällig erzeugt werden, kann in der Spinbox der Grad des Polynoms angegeben werden. Pseudozufällige Polynome werden in dieser Arbeit immer mit den Funktionen der NTL erzeugt. Das irreduzible Polynom `_irredPoly` wird mit der NTL-Funktion `BuildIrred_GF2X(`*long*`)` generiert. Mit dem *Random*-Button werden das irreduzible Polynom und die Koeffizienten a und b generiert. Die Koeffizienten werden mit der Methode `random_GF2X(_irredPoly)` generiert. In kryptographischen Systemen werden oft feste Polynome und Koeffizienten verwendet. Diese können aus dem *Dropdown*-Menü rechts von der Spinbox ausgewählt werden. Die Polynome sind [15] entnommen. Mit dem *OK*-Button wird die gewählte elliptische Kurve über einem Körper der Charakteristik 2 bestätigt und die Operationsfelder werden aktiviert.

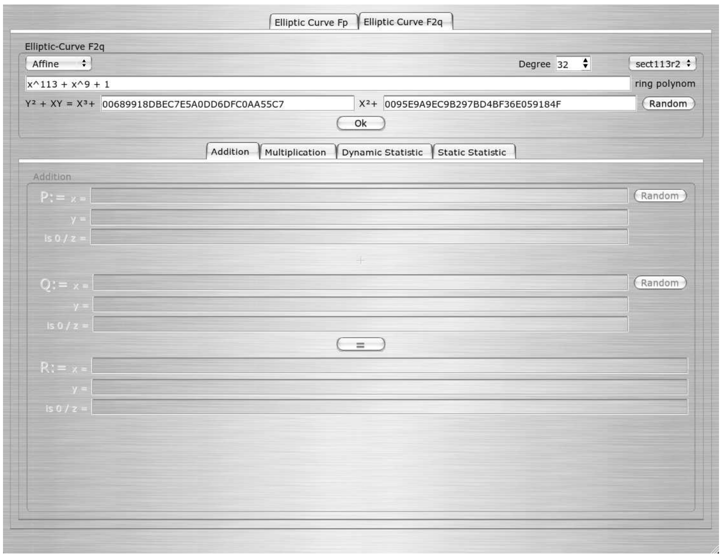

Abbildung 17: Körper und Koordinatensystemwahl

In dem *Additions*-Tab (Abbildung 18) können zwei Punkte auf der gewählten elliptischen Kurve pseudozufällig mit dem *Random*-Button erzeugt werden. Dazu wird eine pseudozufällige x-Koordinate X aus dem Körper $\mathbb{F}_{\texttt{\_irredPoly}}$ mit den Funktionen der NTL generiert und dazu die y-Koordinate berechnet; sollte es keinen Punkt auf der elliptischen Kurve mit der x-Koordinate X geben, so wird ein neues X erzeugt. Dieser Vorgang wird solange wiederholt, bis ein Punkt auf der elliptischen Kurve gefunden ist. Bei der Rückgabe eines Punktes wird jede der beiden möglichen y-Koordinaten mit Wahrscheinlichkeit $\frac{1}{2}$ ausgewählt; ob $y = 0$ gilt wird dabei nicht beachtet. Mit dem "="-Button wird die Summe der beiden Punkte berechnet und dargestellt.

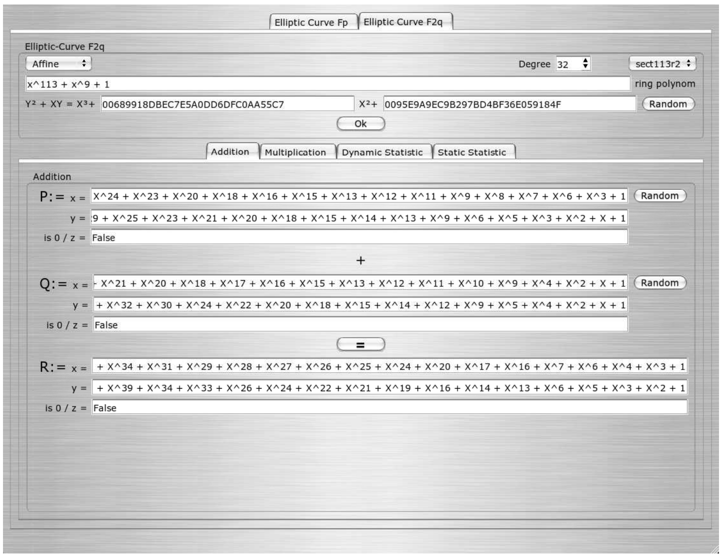

Abbildung 18: Punktaddition

Das *Multiplication*-Tab für elliptische Kurven über Körpern der Charakteristik 2 (Abbildung 19) hat bis auf die Eingabe des Punktes die gleiche Funktionalität wie das *Multiplication*-Tab für elliptische Kurven über Primkörpern. Punkte auf einer Kurve können nicht von Hand eingegeben werden. Ein Skalar und ein Punkt kann pseudozufällig generiert werden. In dem *Dropdown-Menü* kann zwischen den verschiedenen Algorithmen zur skalaren Multiplikation ausgewählt werden. Die Spinbox zur Fenstergröße hat nur dann Einfluss auf die Multiplikation, wenn ein Algorithmus ausgewählt wird, der das Skalar in Fenster aufteilt. Mit dem " = "-Button wird das skalare Produkt berechnet. Mit dem *"Affine Coord."*-Button kann das Ergebnis einer skalaren Multiplikation im projektiven Raum in die affine Koordinatendarstellung projiziert werden.
Bei Aktivierung des Feldes *Number of Operations* werden die aritmetischen Operationen mitgezählt, die für die Punktverdopplungen und Punktadditionen während der Multiplikation auf der elliptischen Kurve benötigt werden. Die arithmetischen Operationen werden nur dann mitgezählt, wenn dieses Feld aktiviert ist.

In dem *"Dynamic Statistic"*-Tab werden skalare Multiplikationen auf pseudozufällig erzeugten elliptischen Kurven über Körpern der Charakteristik 2 berechnet und die Anzahl der Operationen visualisiert (Abbildung 20). In der

Abbildung 19: skalare Multiplikation

Number of tests Spinbox wird angegeben, über wieviele pseudozufällig erzeugte elliptische Kurven und Körper eines Grades, zufällige skalare Multiplikationen berechnet werden. In den *Begin Degree* und *End Degree* Spinboxen wird das Intervall festgelegt, indem die Grade der pseudozufällig erzeugten elliptischen Kurven liegen. In dem *Dropdown*-Menü wird der Algorithmus zur skalaren Multiplikation ausgewählt, über den die Statistik erstellt wird.
Beispiel: Es wird in der *Number of tests* Spinbox der Wert 100, in der *"Begin Degree"* Spinbox 32, in der *"End Degree"* Spinbox 64 und in dem *Dropdown*-Menü *SD-Window* eingestellt. Bei einem Test werden je 100 unterschiedliche elliptische Kurven über unterschiedlichen Körpern der Charakteristik 2 von jedem Grad zwischen 32 und 64 erzeugt. Auf jeder Kurve wird eine pseudozufällige skalare Multiplikation mit dem m-ären Sliding-Window-Multiplizierer berechnet.
Über die Anzahl der unterschiedlichen arithmetischen Operationen und Kurvenoperationen wird der Durchschnitt berechnet und als Punkt im Graph dargestellt. Mit dem *"Start Tests"*-Button wird der Test gestartet.

In dem *"Static Statistic"*-Tab werden viele pseudozufällige skalare Multiplikationen auf einer festen elliptischen Kurve über einem Körper der Charakteristik 2 berechnet und visualisiert (Abbildung 21). In der *"Number of tests"*-Spinbox wird die Anzahl der pseudozufälligen skalaren Multiplikationen eingestellt, die auf der festgelegten elliptischen Kurve berechnet werden. Die skalaren Multiplikationen werden mit den Algorithmen berechnet, die ausgewählt sind. Dabei wird mit jedem der unterschiedlichen Algorithmen dasselbe skalare Produkt berechnet und die Anzahl der arithmetischen Operationen und der Kurvenoperationen als einzelner Punkt dargestellt. Die einzelnen Punkte ergeben eine Punktwolke, die die Streuung der benötigten Operationen visualisiert. Mit dem *"Start Tests"*-Button wird die Statistik erstellt.

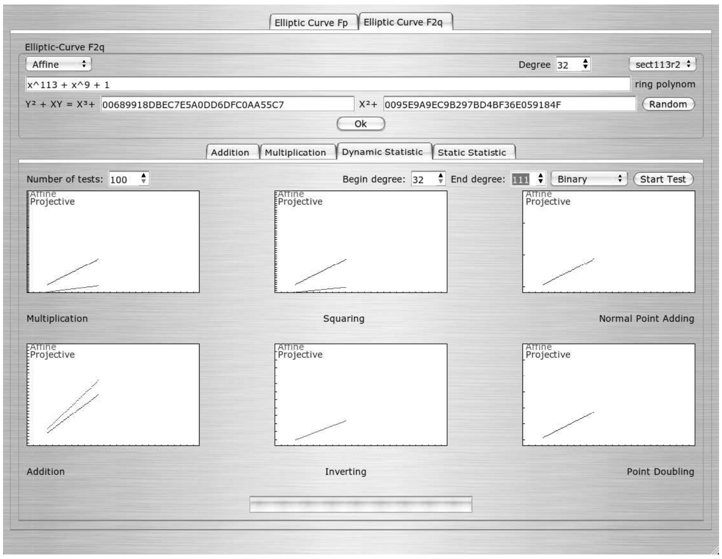

Abbildung 20: Dynamischer Test

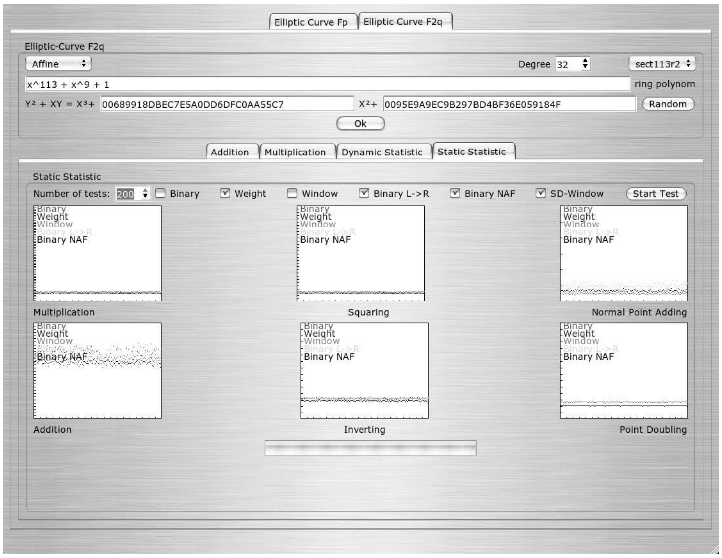

Abbildung 21: Statischer Test

6.3 Modulare Funktionen

6.3.1 Barrett-Reduktion

Für die Bibliothek werden Algorithmen zur modularen Reduktion implementiert. Für den Algorithmus zur Barrett-Reduktion ist die Basis $b = 2$. Durch diese Wahl von b können ganzzahlige Divisionen, Multiplikationen und modulare Reduktionen mit Zweierpotenzen sehr effizient durch Shifts auf einem Computer berechnet werden (Abbildungen 7, 8, 9). Der Algorithmus, wie er in [5] vorgestellt ist, kann nur dann eine Zahl x vollständig im Körper $\mathbb{F}_p$ reduzieren, wenn x kleiner als das Quadrat von p ist. Bei der Implementierung des Algorithmus wird darauf geachtet, dass der Algorithmus für beliebig lange Eingaben funktioniert. Dazu wird der Algorithmus in einer Schleife geschrieben, die p solange quadriert, bis x kleiner als p^n ist. Dann wird $x' \equiv x \pmod{p^n}$ berechnet; x' ist echt kleiner als p^n. Die Schleife wird solange durchlaufen, bis das Ergebnis echt kleiner als p ist.

6.3.2 Montgomery-Reduktion

Es soll eine modulare Reduktion $z \equiv x \pmod{n}$ berechnet werden. Soll diese Reduktion mit der Montgomery-Reduktion berechnet werden, muss eine Basis b gewählt werden, die teilerfremd zu n ist. In der Bibliothek, die erstellt wird, werden mehrere unterschiedliche Versionen der Montgomery-Reduktion implementiert. Eine Implementierung ist geeignet für beliebige Basen und eine ist fest für die Basis 2 implementiert. Beide Versionen benötigen die Parameter $x = (x_{2i}x_{2i-1} \cdots x_1 x_0)_b$ in Montgomeryform, $n = (n_{i-1}n_{i-2} \cdots n_1 n_0)_b$, $R = b^i$ und $n' = -n^{-1} \pmod{R}$. Wenn elliptische Kurven über Primkörpern verwendet werden, kann immer die Implementierung mit der festen Basis 2 eingesetzt werden, da für jede Primzahl $p > 3$ gilt: $GGT(2, p) = 1$. Bei dieser Implementierung muss eine Multiplikation weniger berechnet werden, die sonst im allgemeinen Fall berechnet werden muss. Stattdessen wird nur ein Bit betrachtet, da für den Fall $b = 2$ immer gilt $y_i \equiv y_i \cdot p' \pmod{2}$.
Die beschriebene Implementierung eignet sich für *langzahlarithmetische* Operationen. In der Bibliothek befindet sich eine einfachere Implementierung der Montgomery-Reduktion. Diese ist in [5] beschrieben.

7 Schluss

In dieser Arbeit sind verschiedene Koordinatensysteme zu elliptischen Kurven über Primkörpern und über Körpern der Charakteristik 2 beschrieben. Die Punktoperationen auf diesen Kurven sind analysiert und verglichen. Dabei zeigt sich, dass Punktoperationen auf elliptischen Kurven über Primkörpern und über Körpern der Charakteristik 2 die meisten Bitoperationen benötigen, wenn das affine Koordinatensystem genutzt wird.
Es sind unterschiedliche Algorithmen zur modularen Reduktion gegenübergestellt, wobei sich zeigt, dass eine direkte Implementierung der Barrett-Reduktion für elliptische Kurven über endlichen Körpern gut geeignet ist. Die Montgomery-Reduktion ist in Bitoperationen das schnellere Verfahren, jedoch muss bei die-

sem Algorithmus darauf geachtet werden, dass die Werte in der korrekten Form gehalten werden; dies erhöht die Anzahl der benötigten Operationen stark. Für spätere Arbeiten bietet sich eine genaue Gegenüberstellung der Barrett-Reduktion und der Montgomery-Reduktion in Verbindung mit der Montgomery-Multiplikation und anderen Optimierungen an.
Verschiedene Algorithmen zur skalaren Multiplikation auf elliptischen Kurven sind vorgestellt und miteinander verglichen. Bei der Analyse zeigt sich, dass nicht nur das Hamming-Gewicht, sondern auch die Positionierung der Einsen und Null in der Binärdarstellung einer ganzen Zahl beachtet werden muss.
Die Grundlagen zu einer ECC-Bibliothek in C++ sind geschaffen, die sich für weitere Studienarbeiten anbietet. Algorithmen zur Bestimmung der Ordnung einer elliptischen Kurve und Identitätsfunktionen müssen mindestens noch implementiert werden, damit die Bibliothek für kryptographische Zwecke eingesetzt werden kann. Durch die Modularität kann die Bibliothek, mit einem gut optimierenden Präprozessor, auch auf Systemen mit wenig Speicherplatz eingesetzt werden, da die einzelnen Algorithmen weitestgehend voneinander unabhängig sind.

8 Bibliographie

Literatur

[1] Annette Werner, *Elliptische Kurven in der Kryptographie*, Universität Münster: Springer-Verlag, 2002

[2] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone, *Handbook of Applied Crytography* University of Waterloo (Canada): CRC-Press, 1996

[3] Douglas R. Stinson, *Cryptography - Theorie and Practice - Second Edition*, University of Waterloo (Canada): CRC-Press 2002

[4] Armin Leutbecher, *Zahlentheorie - Eine Einführung in die Algebra*, Technische Universität München: Springer-Verlag 1996

[5] Ian Blake, Gadiel Seroussi, Nigel Smart, *Elliptic Curves in Cryptography*, University Cambridge (Great Britain): Cambridge University Press, 2004

[6] Neal Koblitz, *A Course in Number Theory and Cryptography*, University of Washington, Seattle (USA): Springer-Verlag, 1987

[7] Joachim von zur Gathen, Jürgen Gerhard, *Modern Computer Algebra - zweite Auflage*, Universität Paderborn: Cambridge University Press, 2003

[8] Bjarne Stroustrup, *Die C++-Programmiersprache*, Addison-Wesley, 2000

[9] Thomas Laubrock, *http://www.elliptische-kurven.de*, Internetpublikation, Universität Dortmund: Diplomarbeit, 1999

[10] Marc Joye, Christoph Tymen, *Protections against Differential Analysis for Elliptic Curve Cryptography - An Algebraic Approach*, CHES 2001, Springer-Verlag, 2001 , LNCS , Vol. 2162 , S. 377-390

[11] Lawrence C. Washington, *Elliptic Curves - Number Theory and Cryptography*, CRC-Press, 2003

[12] Annegret Weng, *Der diskrete Logarithmus auf gewöhnlichen elliptischen Kurven mit einem Endomorphismenring kleiner Klassenzahl*, Johann Wolfgang v. Goethe-Universität Frankfurt: Diplomarbeit, 1999

[13] R. Fischlin, *Kryptographische Algorithmen: Effiziente Arithmetik in Z_M und $\mathbb{F}_{p^n}$ (Montgomery-Darstellung, Optimale Normalbasen)*, Internetpublikation:
http://ismi.math.uni-frankfurt.de/schnorr/lecturenotes/montgomery.pdf

[14] The Institute of Electrical and Electronics Engineers, Inc., *IEEE P1363 / D13 (Draft Version 13) - Standard Specifications for Public Key Cryptography*, IEEE Standards Department, 1999

[15] Certicom Research, *SEC 2: Recommended Elliptic Curve Domain Parameters*, Internetpublikation, 2000, Version 1.0
http://www.secg.org/download/aid-386/sec2_final.pdf

[16] Jean-Sébastian Coron, *Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems*, CHES 1999, Springer-Verlag, 1999 , LNCS , Vol. 1717 , S. 292-302

[17] H. Park, K. Park, Y. Cho, Analysis of the Variable Length Nonzero Window Method for Exponentiation,Seoul National University(Korea): Computers and Mathematics with Applications, 1999, Vol. 37, S. 21-29

[18] Martin Otto, *Brauer Addition-Subtraction Chains*, Universität Paderborn: Diplomarbeit, 2001

Titelbild: [7]
http://www-math.uni-paderborn.de/mca/addition.png

9 Anhang

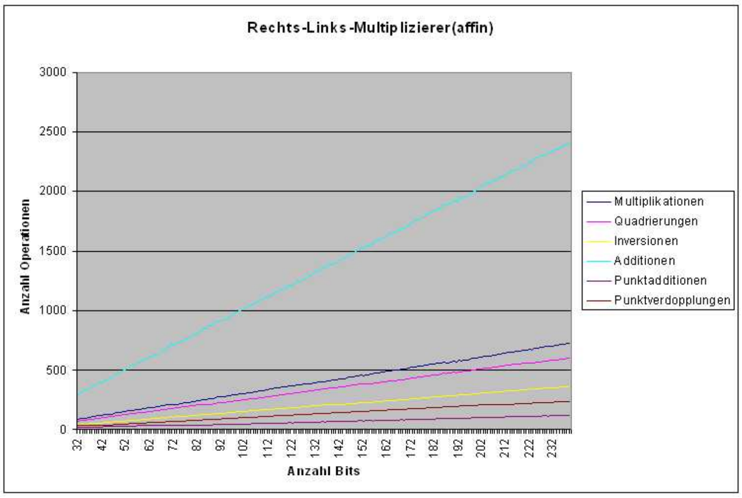

Abbildung 22: Operationen auf elliptischer Kurve in affiner Darstellung bei skalarer Multiplikation durch wiederholtes Verdoppeln von rechts nach links

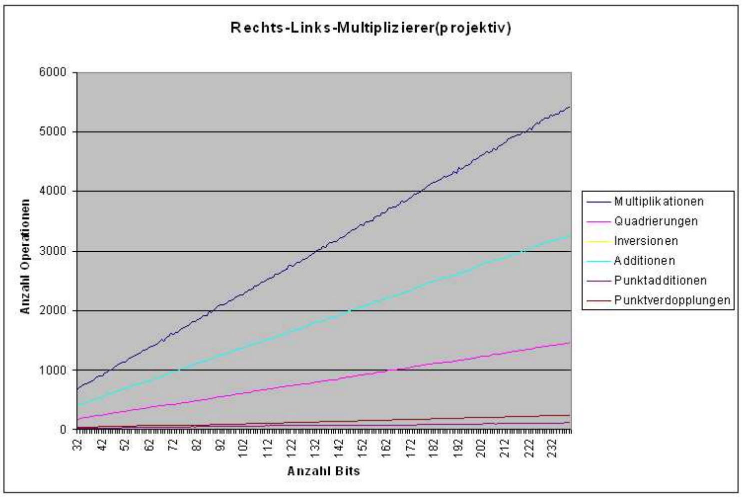

Abbildung 23: Operationen auf elliptischer Kurve in projektiver Darstellung bei skalarer Multiplikation durch wiederholtes Verdoppeln von rechts nach links

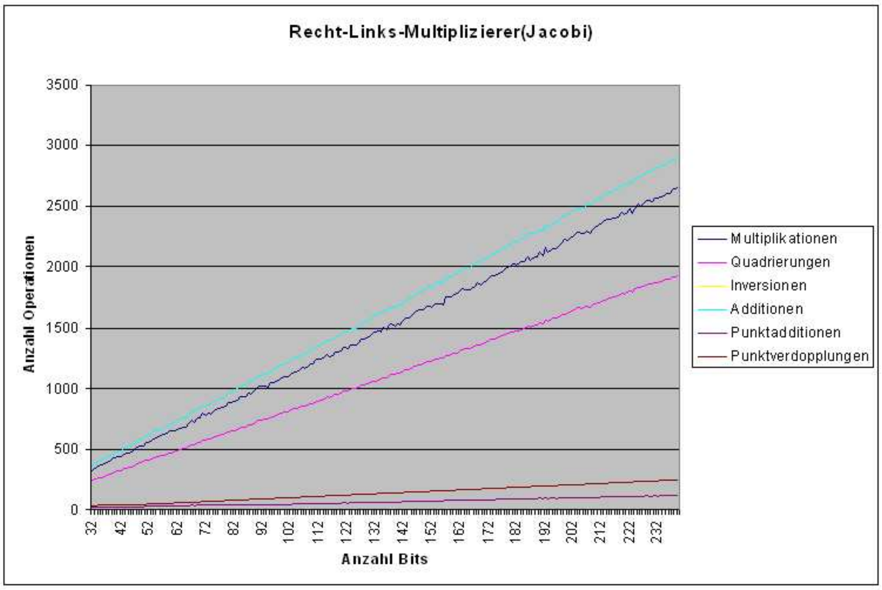

Abbildung 24: Operationen auf elliptischer Kurve in Jacobidarstellung bei skalarer Multiplikation durch wiederholtes Verdoppeln von rechts nach links

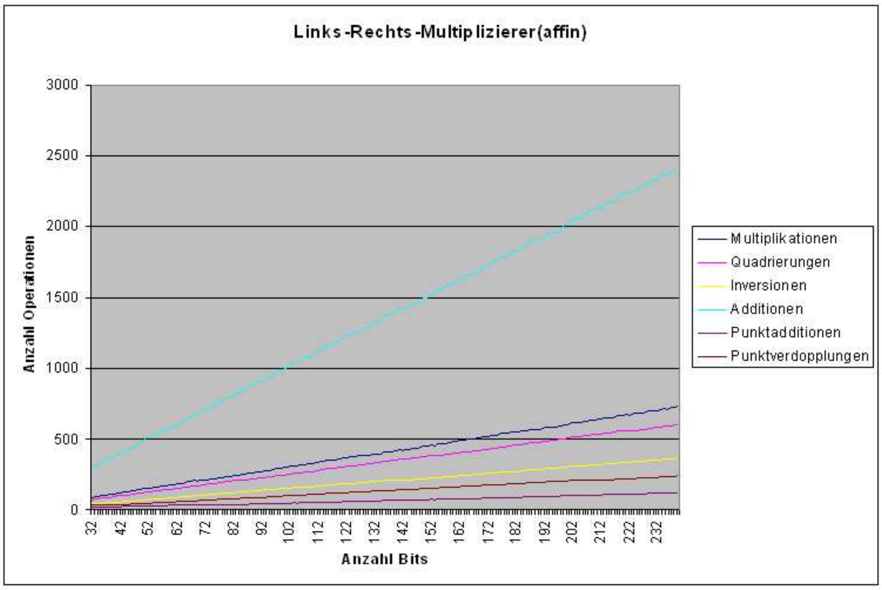

Abbildung 25: Operationen auf elliptischer Kurve in affiner Darstellung bei skalarer Multiplikation durch wiederholtes Verdoppeln von links nach rechts

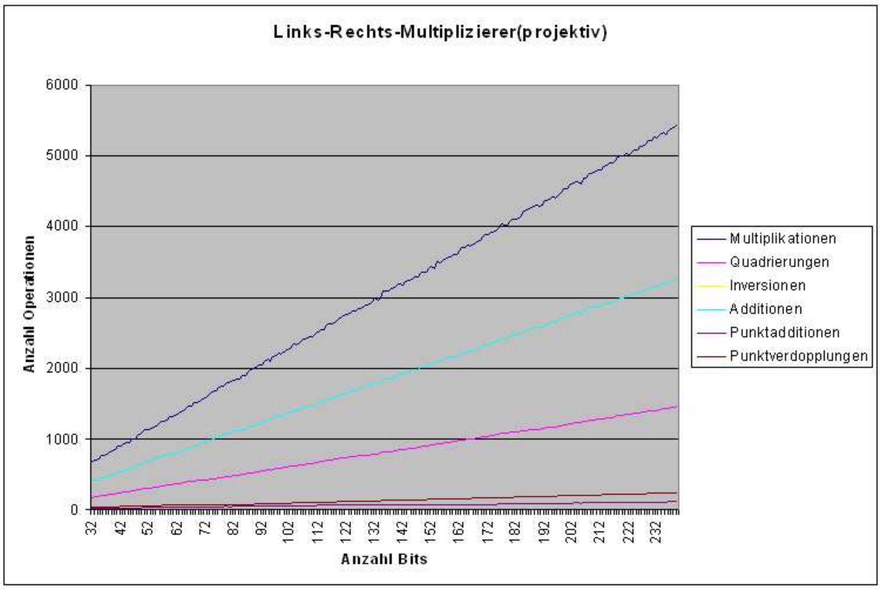

Abbildung 26: Operationen auf elliptischer Kurve in projektiver Darstellung bei skalarer Multiplikation durch wiederholtes Verdoppeln von links nach rechts

Abbildung 27: Operationen auf elliptischer Kurve in Jacobidarstellung bei skalarer Multiplikation durch wiederholtes Verdoppeln von links nach rechts

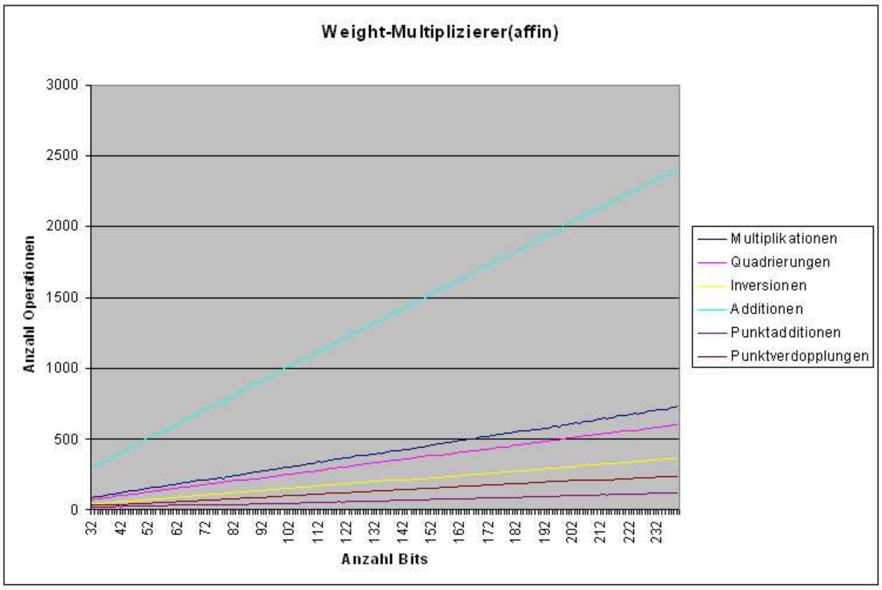

Abbildung 28: Operationen auf elliptischer Kurve in affiner Darstellung bei skalarer Multiplikation durch Weight-Multiplizierer

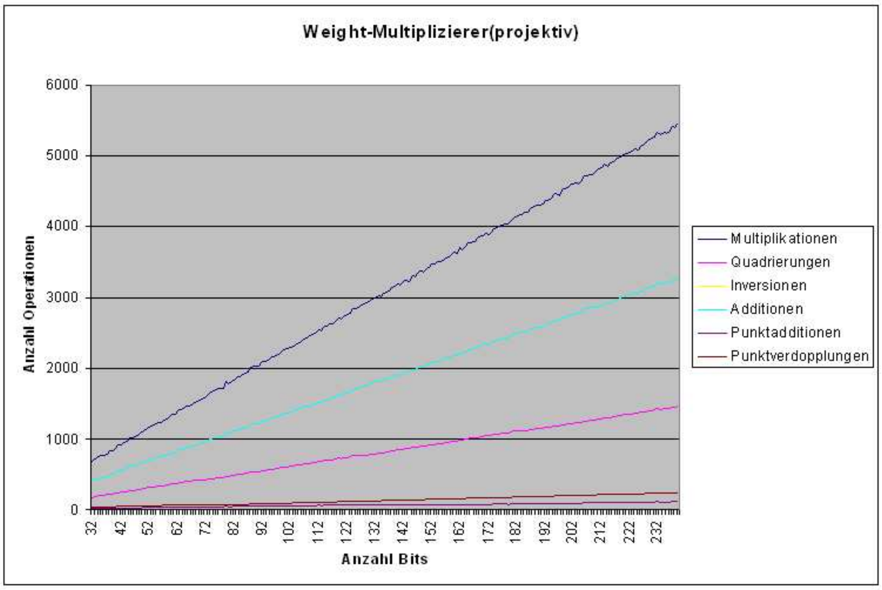

Abbildung 29: Operationen auf elliptischer Kurve in projektiver Darstellung bei skalarer Multiplikation durch Weight-Multiplizierer

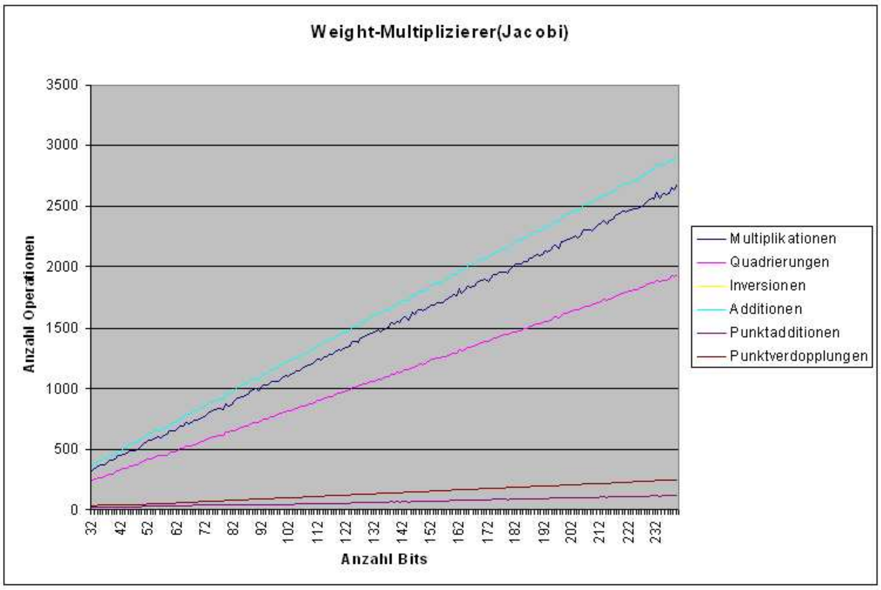

Abbildung 30: Operationen auf elliptischer Kurve in Jacobidarstellung bei skalarer Multiplikation durch Weight-Multiplizierer

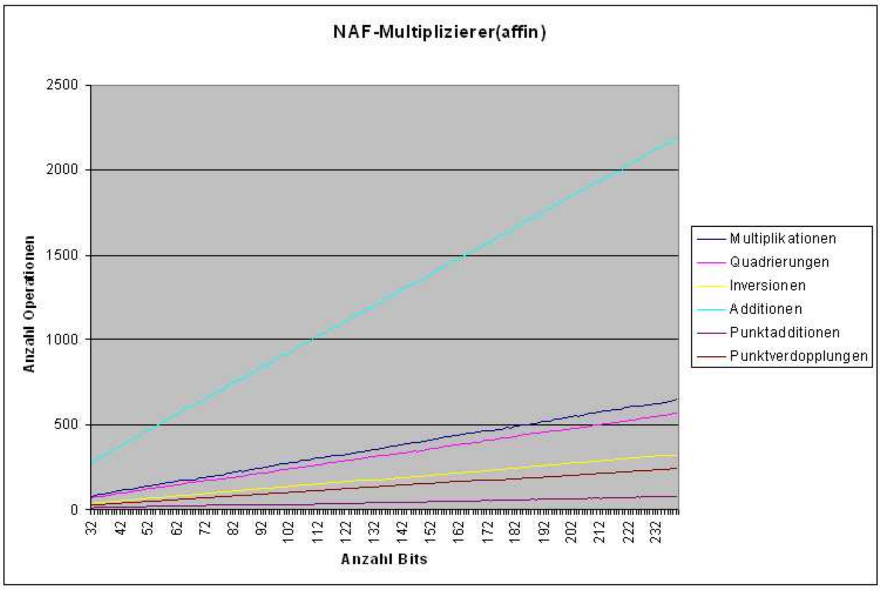

Abbildung 31: Operationen auf elliptischer Kurve in affiner Darstellung bei skalarer Multiplikation durch NAF-Multiplizierer

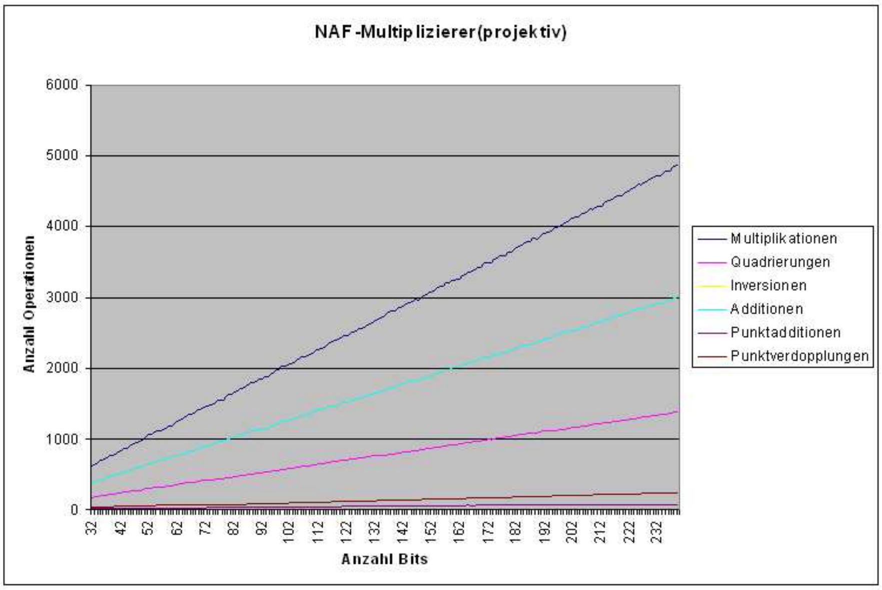

Abbildung 32: Operationen auf elliptischer Kurve in projektiver Darstellung bei skalarer Multiplikation durch NAF-Multiplizierer

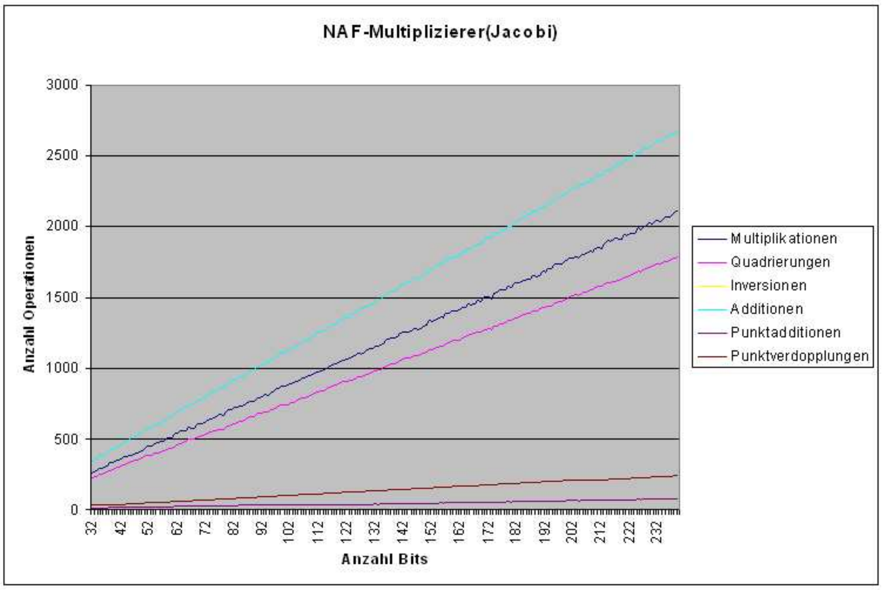

Abbildung 33: Operationen auf elliptischer Kurve in Jacobidarstellung bei skalarer Multiplikation durch NAF-Multiplizierer

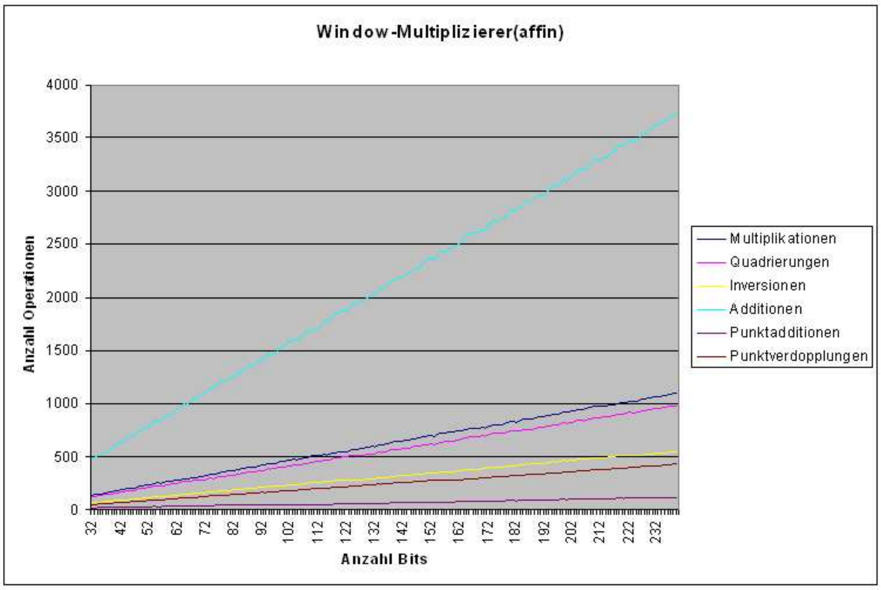

Abbildung 34: Operationen auf elliptischer Kurve in affiner Darstellung bei skalarer Multiplikation durch Window-Multiplizierer, bei Fenstergröße 4

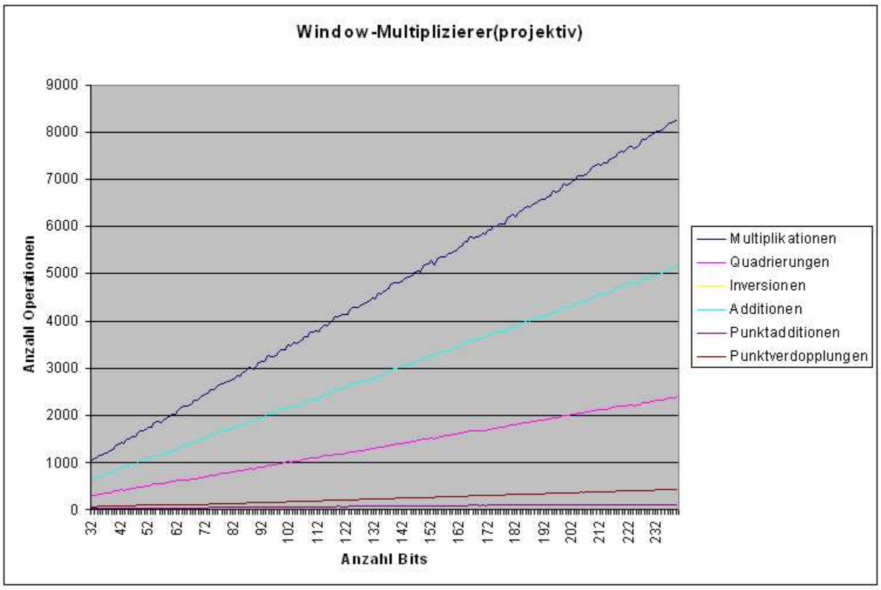

Abbildung 35: Operationen auf elliptischer Kurve in projektiver Darstellung bei skalarer Multiplikation durch Window-Multiplizierer, bei Fenstergröße 4

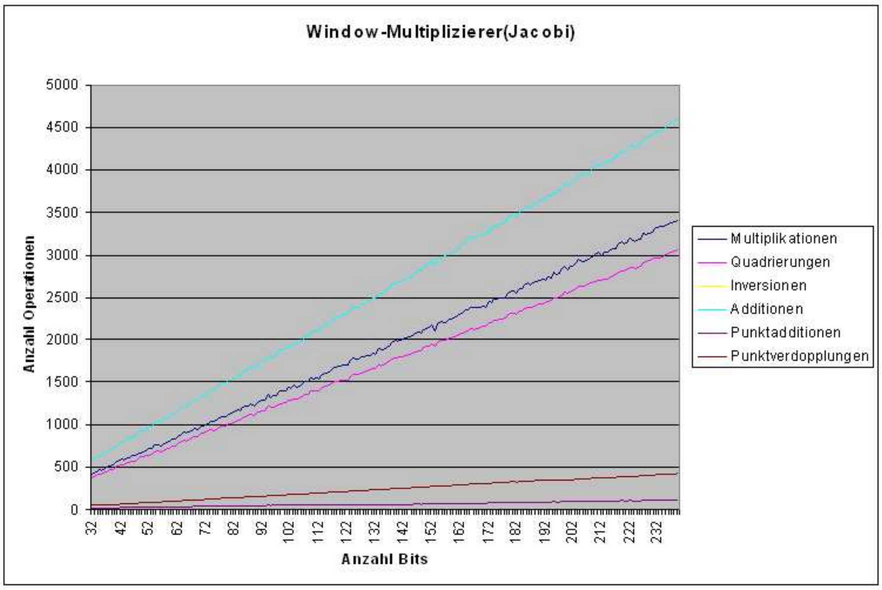

Abbildung 36: Operationen auf elliptischer Kurve in Jacobidarstellung bei skalarer Multiplikation durch Window-Multiplizierer, bei Fenstergröße 4

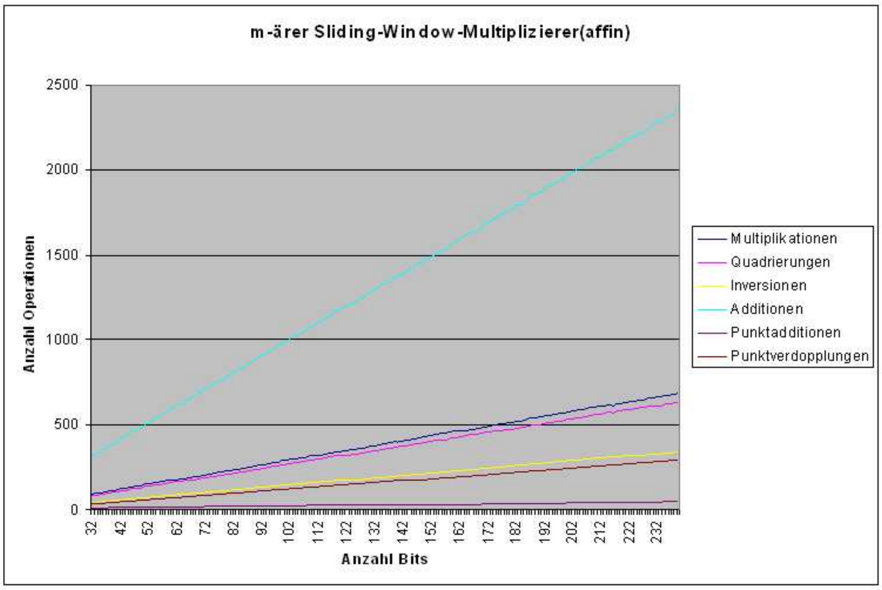

Abbildung 37: Operationen auf elliptischer Kurve in affiner Darstellung bei skalarer Multiplikation durch m-ären Sliding-Window-Multiplizierer, bei Fenstergröße 4

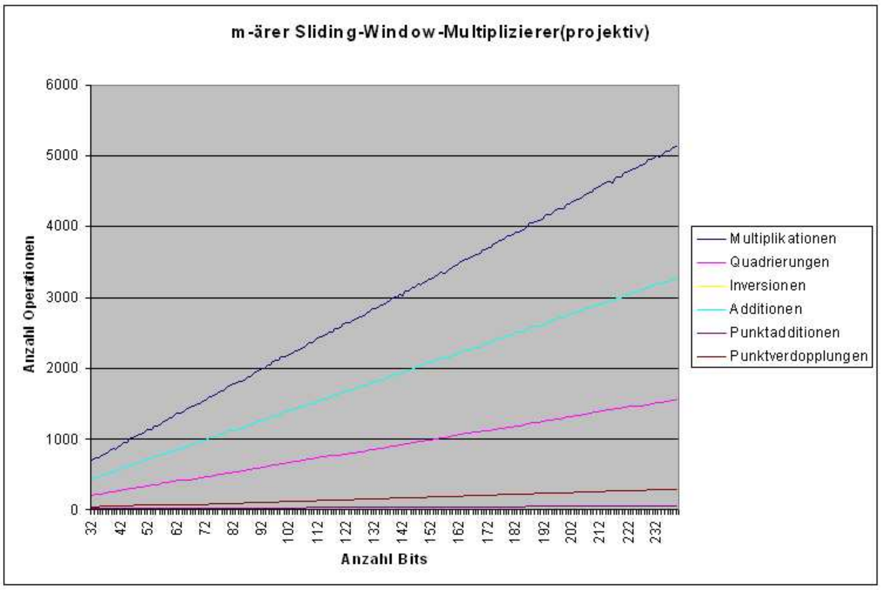

Abbildung 38: Operationen auf elliptischer Kurve in projektiver Darstellung bei skalarer Multiplikation durch m-ären Sliding-Window-Multiplizierer, bei Fenstergröße 4

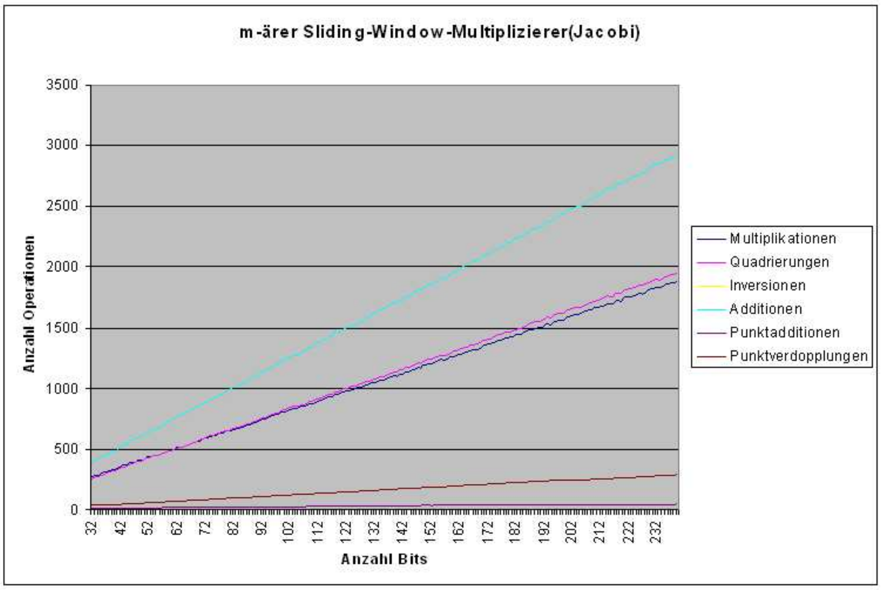

Abbildung 39: Operationen auf elliptischer Kurve in Jacobidarstellung bei skalarer Multiplikation durch m-ären Sliding-Window-Multiplizierer, bei Fenstergröße 4